35th International Symposium on Algorithms and Computation

ISAAC 2024, December 8–11, 2024, Sydney, Australia

Edited by

Julián Mestre
Anthony Wirth

LIPIcs – Vol. 322 – ISAAC 2024 www.dagstuhl.de/lipics

Editors

Julián Mestre
School of Computer Science, The University of Sydney, Australia
julian.mestre@sydney.edu.au

Anthony Wirth
School of Computer Science, The University of Sydney, Australia
anthony.wirth@sydney.edu.au

ACM Classification 2012
Theory of computation → Design and analysis of algorithms; Theory of computation → Computational complexity and cryptography; Theory of computation → Randomness, geometry and discrete structures

ISBN 978-3-95977-354-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-354-6.

Publication date
December, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ISAAC.2024.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume comprises the papers presented at the 35th International Symposium on Algorithms and Computation (ISAAC 2024), which was held in Sydney, Australia on 8–11 December 2024, organized by The University of Sydney. ISAAC 2024 provided a forum for researchers working in the areas of algorithms, theory of computation, and computational complexity.

Of the 193 submissions to ISAAC 2024, the 44 members of the program committee (PC), listed below, curated a program of 56 technical papers. Each submission received at least three reviews, with several written by external reviewers invited by the program committee, also listed below. We are most grateful for the careful and considered reviews of the PC members and external reviewers, as well as the attentive discussion and decisions of the PC. We are delighted to present the technical program to you on their behalf.

The program committee selected the following papers as the recipients of the ISAAC 2024 Best Paper and Best Student Paper Awards.

Best Paper Vadim Lozin, Barnaby Martin, Sukanya Pandey, Daniel Paulusma, Mark Siggers, Siani Smith and Erik Jan van Leeuwen. *Complexity Framework for Forbidden Subgraphs II: Edge Subdivision and the "H"-graphs.*

Best Student Paper Koustav Bhanja. *Optimal Sensitivity Oracle for Steiner Mincut.*

The Symposium welcomed three invited presentations, by Ravi Kumar (Google), Olga Ohrimenko (U. Melbourne), and Barna Saha (UCSD). We are very grateful to have such quality presenters, and are pleased to include their abstracts below. The week prior to the Symposium, The University of Sydney hosted the four-day summer school on "Recent Trends in Algorithms" for students. We thank The University of Sydney and Google for their support of ISAAC 2024.

December 2024

Julián Mestre and Anthony Wirth

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Adi Rosén (CNRS & Université Paris Cité)

Akira Suzuki (Tohoku University)

Amit Kumar (IIT Delhi)

Anthony Wirth (University of Sydney) [co-chair]

Catherine Greenhill (UNSW)

Chien-Chung Huang (CNRS & ENS Ulm)

Davide Bilò (University of L'Aquila)

Diptarka Chakraborty (National University of Singapore)

Edith Elkind (University of Oxford)

Eunjin Oh (POSTECH)

Fabrizio Frati (Università Roma Tre)

Guochuan Zhang (Zhejiang University)

Hanna Sumita (Tokyo Institute of Technology)

Ho-Lin Chen (National Taiwan University)

Holger Dell (Goethe-Universität Frankfurt)

Hyung-Chan An (Yonsei University)

Ioana O. Bercea (KTH)

Jian Li (Tsinghua University)

Jittat Fakcharoenphol (Kasetsart University)

Julián Mestre (University of Sydney) [co-chair]

Loukas Georgiadis (University of Ioannina)

Marcos Kiwi (University of Chile)

Mark de Berg (TU Eindhoven)

Matthias Mnich (TU Hamburg)

Meng-Tsung Tsai (Academia Sinica)

Michael Lampis (Université Paris Dauphine)

Nicole Megow (University of Bremen)

Paloma T. Lima (IT University of Copenhagen)

Paul Spirakis (University of Liverpool, UK)

Ragesh Jaiswal (IIT Delhi)

Saket Saurabh (Institute of Mathematical Sciences)

Sampson Wong (University of Copenhagen)

Samson Zhou (Texas A&M University)

Sándor Fekete (TU Braunschweig)

Santhoshini Velusamy (TTIC)

Shi Li (Nanjing University)

Shunhua Jiang (Columbia University)

Stefan Funke (University of Stuttgart)

Takehiro Ito (Tohoku University)

Thomas Erlebach (Durham University)

Troy Lee (University of Technology Sydney)

Vaggos (Evangelos) Chatziafratis (UCSC)

Wing-Kai Hon (National Tsing Hua University)

Yi-Jun Chang (National University of Singapore)

Yiding Feng (HKUST IEDA)

Zhiyi Huang (University of Hong Kong)

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Aaron Putterman

Abheek Ghosh

Adam Karczmarz

Adarsh Srinivasan

Aditya Anand

Aleksandr M. Kazachkov

Aleksandr Popov

Alessandro Straziota

Alessia Di Fonso

Alessio Conte

Alexander Kulikov

Alexandra Lassota

Ameet Gadekhar

Amer Mouawad

Amir Nikabadi

André Nusser

André van Renssen

Andrea Munaro

Aniket Basu Roy

Anna Lubiw

Anna Mpanti

Antonios Symvonis

Aravind Thyagarajan

Argyrios Deligkas

Argyris Oikonomou

Arnab Ganguly

Arnold Filtser

Arsen Vasilyan

Athanasios Konstantinidis

Atul Mantri

Benedikt Kolbe

Benjamin Aram Berendsohn

Bento Natura

Biaoshuai Tao

Binghui Peng

Bingkai Lin

Blake Holman

Bo Li

Boaz Patt-Shamir

Carlos Alegría

Carlos Seara

Carolina Lucía Gonzalez

Casper Rysgaard

Ce Jin

Chen Wang

Chenyang Xu

Chetan Gupta

Ching-Chi Lin

Christian Coester

Christian Rieck

Clément Canonne

Csaba Toth

Daniel Gibney

Daniel Grier

Daniel Noble

Daniel Paul-Pena

Daniel Vaz

David Eppstein

Daya Gaur

Debajyoti Bera

Dillon Mayhew

Dimitris Christou

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dionysios Kefallinos
Dominik Krupke
Dong Deng
Duncan Adamson
Eric Brandwein
Erik Jan van Leeuwen
Evangelos Kosinas
Fabiano Oliveira
Fabrizio Grosso
Farbod Ekbatani
Felicia Lucke
Felix Lebrat
Flavia Bonomo
Florent Foucaud
Frederick Stock
Gabriele Di Stefano
Gaia Carenini
Giordano Da Lozzo
Giuseppe Romana
Gregory Kucherov
Grzegorz Kwasniewski
Guilherme D. Da Fonseca
Guilherme de Castro Mendes Gomes
Guillaume Ducoffe
Haoqiang Huang
Hendrik Molter
Hoa Vu
Hsiang-Hsuan Liu
Hua Chen
Hugo Gilbert
Hung Hoang
Hung Le
Hung-Lung Wang

Ignaz Rutter
Ilie Sarpe
Irfansha Shaik
Ivan Bliznets
Jack Stade
Jan Kaiser
Janani Sundaresan
Janka Chlebikova
Jatin Yadav
Jean-Lou De Carufel
Jelle Oostveen
Jie Xue
Jiehua Chen
João F. Doriguello
Joseph Mitchell
Josh Alman
Jules Wulms
Jun Kawahara
Jungho Ahn
Junichi Teruyama
Justin Ward
Karolina Okrasa
Katrin Casel
Kazuhiro Kurita
Kelin Luo
Kirill Simonov
Konstantinos Tsakalidis
Kostas Tsichlas
Kristóf Bérczi
Kushagra Chatterjee
Kyungjin Cho
László Kozma
Laure Morelle

Leah Epstein

Leszek Gasieniec

Lin Chen

Ling-Ju Hung

Liren Shan

Lorenzo Beretta

Lorenzo Carfagna

Louis Esperet

Luciano Gualà

Magnus Wahlström

Manoj Gupta

Manoj Gupta

Manolis Vasilakis

Marc Pfetsch

Marco D'Elia

Marios Mavronicolas

Markus Chimani

Markus Hecher

Markus L. Schmid

Martin Bullinger

Martin Nöllenburg

Massimo Lauria

Mathieu Mari

Matthias Bentert

Maximilian Stahlberg

Meng He

Mengqian Zhang

Michael Payne

Michael Perk

Mikhail Isaev

Mikkel Abrahamsen

Milan Mosse

Miriam Münch

Mohammad Ali Abam

Mohammad Reza Aminian

Mordecai J. Golin

Moritz Muehlenthaler

N.S. Narayanaswamy

Nairen Cao

Nate Veldt

Nathan Flaherty

Neelima Gupta

Nicholas Teh

Niclas Boehmer

Nicole Wein

Nidhi Purohit

Nikhil Balaji

Nikhil Mande

Nils Morawietz

Nina Klobas

Nitin Saurabh

Noah Fleming

Noah Singer

Noel Arteche

Noleen Köhler

Omkar Bhalerao

Ondřej Suchý

Pablo Rotondo

Pamela Fleischmann

Pascal Baßler

Pascal Kunz

Patrizio Angelini

Peiyuan Liu

Peter Davies-Peck

Peter Kramer

Petr Golovach

Philipp Kindermann
Pierre Fraigniaud
Po-An Chen
Pooja Kulkarni
Prafullkumar Tale
Prantar Ghosh
Puping Jiang
Quanquan Liu
Rahul Shah
Rajni Dabas
Ramin Kosfeld
Razvan Barbulescu
René Sitters
Renfei Zhou
Rin Saito
Robert Andrews
Romain Cosson
Ron Safier
Ruizhe Zhang
Ryoga Mahara
Sabine Cornelsen
Sabine Storandt
Sabyasachi Basu
Sandeep Silwal
Sanjana Dey
Sarita de Berg
Sasha Rubin
Satyabrata Jana
Sebastian Brandt
Seeun William Umboh
Shang-En Huang
Shaofeng Jiang
Shayan Oveis Gharan
Shengwei Zhou
Shinwoo An
Shuai Shao
Shuichi Hirahara
Simon J. Puglisi
Simon Meierhans
Sorrachai Yingchareonthawornchai
Sriram Bhyravarapu
Stanisław Gawiejnowicz
Stefan Walzer
Stefano Leucci
Stephen Kobourov
Stoyan Dimitrov
Sudatta Bhattacharya
Sumanta Ghosh
Sungmin Kim
Susanna Caroppo
Sushmita Gupta
T-H. Hubert Chan
Ta-Wei Tu
Takahiro Suzuki
Talya Eden
Tatsuhiro Suga
Tatsuya Gima
Ted Pyne
Tesshu Hanaka
Thatchaphol Saranurak
Thekla Hamm
Théo Pierron
Thijs van der Horst
Thirupathaiah Vasantam
Tian Bai
Torsten Mütze

Travis Gagie

Tsz Chiu Kwok

Ulrich Pferschy

Uri Zwick

Vaishali Suriyanaryanan

Valentin Polishchuk

Viktor Zamaraev

Vincent Chau

Vittorio Bilò

Vladimir Podolskii

Wei Tang

Wei Yu

Xujin Chen

Yakov Nekrich

Yaniv Sadeh

Yanlin Chen

Yasuaki Kobayashi

Yasushi Kawase

Yatong Chen

Yecheng Xue

Yican Sun

Yinzhan Xu

Yiyuan Luo

Yong Zhang

Yoshio Okamoto

Yota Otachi

Young-San Lin

Yuan Sha

Yuhao Zhang

Yuki Takeuchi

Yuma Tamura

Zack Jorquera

Zhenwei Liu

Zhihao Gavin Tang

Zihan Tan

Zijin Huang

Algorithmic Problems in Discrete Choice

Ravi Kumar
Google, Mountain View, CA, USA

Abstract

In discrete choice, a user selects one option from a finite set of available alternatives, a process that is crucial for recommendation systems applications in e-commerce, social media, search engines, etc. A popular way to model discrete choice is through Random Utility Models (RUMs). RUMs assume that users assign values to options and choose the one with the highest value from among the available alternatives. RUMs have become increasingly important in the Web era; they offer an elegant mathematical framework for researchers to model user choices and predict user behavior based on (possibly limited) observations. While RUMs have been extensively studied in behavioral economics and social sciences, many basic algorithmic tasks remain poorly understood. In this talk, we will discuss various algorithmic and learning questions concerning RUMs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Mathematics of computing → Probability and statistics; Information systems → Data mining

Keywords and phrases discrete choice theory, random utility models, user behavior

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.1

Category Invited Talk

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 1; pp. 1:1–1:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Data Privacy: The Land Where Average Cases Don't Exist and Assumptions Quickly Perish

Olga Ohrimenko
The University of Melbourne, Australia

Abstract

Machine learning on personal and sensitive data raises serious privacy concerns and creates potential for inadvertent information leakage (e.g., extraction of private messages or images from generative models). However, incorporating analysis of such data in decision making can benefit individuals and society at large (e.g., in healthcare). To strike a balance between these two conflicting objectives, one must ensure that data analysis with strong confidentiality guarantees is deployed and securely implemented.

Differential privacy (DP) is emerging as a leading framework for analyzing data while maintaining mathematical privacy guarantees. Although it has seen some real-world deployment (e.g., by Apple, Microsoft, and Google), such instances remain limited and are often constrained to specific scenarios. Why?

In this talk, I argue that part of the challenge lies in the assumptions DP makes about its deployment environment. By examining several DP systems and their assumptions, I demonstrate how private information can be extracted using, for example, side-channel information or the ability to rewind system's state. I then give an overview of efficient algorithms and protocols to realize these assumptions and ensure secure deployment of differential privacy.

2012 ACM Subject Classification Security and privacy → Systems security; Security and privacy → Privacy-preserving protocols; Security and privacy → Hardware attacks and countermeasures

Keywords and phrases Differential privacy, side-channel attacks, trusted execution environment, privacy budget, state continuity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.2

Category Invited Talk

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Role of Structured Matrices in Fine-Grained Algorithm Design

Barna Saha
Department of Computer Science and Engineering & Data Science,
University of California San Diego, La Jolla, CA, USA

Abstract

Fine-grained complexity attempts to precisely determine the time complexity of a problem and has emerged as a guide for algorithm design in recent times. Some of the central problems in fine-grain complexity deals with computation of distances. For example, computing all pairs shortest paths in a weighted graph, computing edit distance between two sequences or two trees, and computing distance of a sequence from a context free language. Many of these problems reduce to computation of matrix products over various algebraic structures, predominantly over the (min,+) semiring. Obtaining a truly subcubic algorithm for (min,+) product is one of the outstanding open questions in computer science.

Interestingly many of the aforementioned distance computation problems have some additional structural properties. Specifically, when we perturb the inputs slightly, we do not expect a huge change in the output. This simple yet powerful observation has led to better algorithms for many problems for which we were able to improve the running time after several decades. This includes problems such as the Language Edit Distance, RNA folding, and Dyck Edit Distance. Indeed, this structure in the problem leads to matrices that have the Lipschitz property, and we gave the first truly subcubic time algorithm for computing (min,+) product over such Lipschitz matrices. Follow-up work by several researchers obtained improved bounds for monotone matrices, and for (min,+) convolution under similar structures leading to improved bounds for a series of optimization problems. These result in not just faster algorithms for exact computation but also for approximation algorithms. In particular, we show how fast (min,+) product computation over monotone matrices can lead to better additive approximation algorithms for computing all pairs shortest paths on unweighted undirected graphs, leading to improvements after twenty four years.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques

Keywords and phrases Fine-Grained Complexity, Fast Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.3

Category Invited Talk

Funding *Barna Saha*: Partially supported by NSF grants 1652303, 1909046, 2112533, and HDR TRIPODS Phase II grant 2217058.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 3; pp. 3:1–3:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Minimum Plane Bichromatic Spanning Trees

Hugo A. Akitaya
Miner School of Computer & Information Sciences, University of Massachusetts, Lowell, MA, USA

Ahmad Biniaz
School of Computer Science, University of Windsor, Canada

Erik D. Demaine
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Linda Kleist
Institute of Computer Science, Universität Potsdam, Germany

Frederick Stock
Miner School of Computer & Information Sciences, University of Massachusetts, Lowell, MA, USA

Csaba D. Tóth
Department of Mathematics, California State University Northridge, Los Angeles, CA, USA
Department of Computer Science, Tufts University, Medford, MA, USA

Abstract

For a set of red and blue points in the plane, a *minimum bichromatic spanning tree* (MinBST) is a shortest spanning tree of the points such that every edge has a red and a blue endpoint. A MinBST can be computed in $O(n \log n)$ time where n is the number of points. In contrast to the standard Euclidean MST, which is always *plane* (noncrossing), a MinBST may have edges that cross each other. However, we prove that a MinBST is quasi-plane, that is, it does not contain three pairwise crossing edges, and we determine the maximum number of crossings.

Moreover, we study the problem of finding a *minimum plane bichromatic spanning tree* (MinPBST) which is a shortest bichromatic spanning tree with pairwise noncrossing edges. This problem is known to be NP-hard. The previous best approximation algorithm, due to Borgelt et al. (2009), has a ratio of $O(\sqrt{n})$. It is also known that the optimum solution can be computed in polynomial time in some special cases, for instance, when the points are in convex position, collinear, semi-collinear, or when one color class has constant size. We present an $O(\log n)$-factor approximation algorithm for the general case.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Bichromatic Spanning Tree, Minimum Spanning Tree, Plane Tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.4

Funding *Hugo A. Akitaya*: Research supported by the NSF award CCF-2348067.
Ahmad Biniaz: Research supported by NSERC.
Csaba D. Tóth: Research supported in part by the NSF award DMS-2154347.

Acknowledgements This work was initiated at the Eleventh Annual Workshop on Geometry and Graphs, held at the Bellairs Research Institute in Holetown, Barbados in March 2024. The authors thank the organizers and the participants.

1 Introduction

Computing a minimum spanning tree (MST) in a graph is a well-studied problem. There exist many algorithms for this problem, among which one can mention the celebrated Kruskal's algorithm [37], Prim's algorithm [41], and Borůvka's algorithm [22]. The running time of

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 4; pp. 4:1–4:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

these algorithms depends on the number of vertices and edges of the input graph. For geometric graphs, where the vertices are points in the plane, their running time depends only on the number of vertices.

For a set S of n points in the plane a Euclidean MST (i.e., an MST of the complete graph on S with straight-line edges and Euclidean edge weights) can be computed in $O(n \log n)$ time. When the points of S are colored by two colors, say red and blue, and every edge is required to have a red and a blue endpoint, then a spanning tree is referred to as a *bichromatic spanning tree*. A *minimum bichromatic spanning tree* (MinBST) is a bichromatic spanning tree of minimum total edge length. A MinBST on S can be computed in $O(n \log n)$ time [18]. When the points are collinear (all lie on a straight line) and are given in sorted order along the line this problem can be solved in linear time [15].

We say that two line segments *cross* if they share an interior point; this configuration is called a *crossing*. A tree is called *plane* if its edges are pairwise noncrossing. The standard Euclidean MST is always plane. This property is ensured by the triangle inequality, because the tree can be made shorter by replacing any two crossing edges with two noncrossing edges. The noncrossing property does not necessarily hold for a MinBST, see Figure 1 for an example. Two crossing edges in this example cannot be replaced with two noncrossing edges because, otherwise, we would either introduce monochromatic edges (that connect points of the same color) or disconnect the tree into two components.

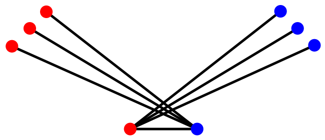


Figure 1 A bicolored point set and its minimum bichromatic spanning tree (MinBST).

Edge crossings in geometric graphs are usually undesirable as they could lead to unwanted situations such as collisions in motion planning, inconsistency in VLSI layout, and interference in wireless networks. They are also undesirable in the context of graph drawing and network visualization. Therefore, it is natural to ask for a *minimum plane bichromatic spanning tree* (MinPBST), a bichromatic spanning tree that is noncrossing and has minimum total edge length. Borgelt et al. [21] proved that the problem of finding a MinPBST is NP-hard. They also present a polynomial-time approximation algorithm with approximation factor $O(\sqrt{n})$.

In this paper we study the MinBST and MinPBST problems from combinatorial and computational points of view. First we present an approximation algorithm, with a better factor, for the MinPBST problem. Then we prove some interesting structural properties of the MinBST.

1.1 Related work

Problems related to bichromatic objects (such as points and lines) have been actively studied in computational geometry, for instance, the problems related to bichromatic intersection [4, 24, 25, 38], bichromatic separation [9, 11, 14, 16, 27], and noncrossing bichromatic connection [1, 2, 17, 19, 21, 32, 34, 36]. We refer the interested reader to the survey by Kaneko and Kano [35].

The $O(\sqrt{n})$-approximation algorithm of Borgelt et al. [21] for the MinPBST problem lays a $(\sqrt{n} \times \sqrt{n})$-grid over the points, then identifies a subset of grid cells as *core* regions and computes their Voronoi diagram, then builds a tree inside each Voronoi cell, and finally combines the trees.

Let ρ_n be the supremum ratio of the length of MinPBST to the length of MinBST over all sets of n bichromatic points. Grantson et al. [29] show that $3/2 \leq \rho_n \leq n$ for all $n \geq 4$; and ask whether the upper bound can be improved. It is easily seen from the algorithm of Borgelt et al. [21] that $\rho_n \leq O(\sqrt{n})$ because the planarity of the optimal solution is not used in the analysis of the approximation ratio – indeed the analysis would work even with respect to the MinBST.

Some special cases of the MinPBST problem can be solved to optimality in polynomial time. For instance, the problem can be solved in $O(n^2)$ time when points are collinear [15], in $O(n^3)$ time when points are in convex position [21], in $O(n^5)$ time when points are semi-collinear (points in one color class are on a line and all other points are on one side of the line) [17], and in $n^{O(k^5)}$ time when one color class has k points for some constant k [21].

One might wonder if a greedy strategy could achieve a better approximation ratio. A modified version of Kruskal's algorithm, that successively adds a shortest bichromatic edge that creates neither a cycle nor a crossing, is referred to as the *greedy algorithm* [21, 29]. This algorithm, as noted in [21, Figure 1], does not always return a planar bichromatic tree (it does not always terminate: there may be a point of one color that cannot see any point of the opposite color).

Abu-Affash et al. [2] studied the *bottleneck* version of the plane bichromatic spanning tree problem where the goal is to minimize the length of the longest edge. They prove that this problem is NP-hard, and present an $8\sqrt{2}$-approximation algorithm.

1.2 Quasi-planarity

Quasi-planarity is a measure of the proximity of an (abstract or geometric) graph to planarity. For an integer $k \geq 2$, a graph is called *k-quasi-planar* if it can be drawn in the plane such that no k edges pairwise cross. By this definition, a planar graph is 2-quasi-planar. A 3-quasi-planar graph is also called *quasi-planar*. Problems on k-quasi-planarity are closely related to Turán-type problems on the intersection graph of line segments in the plane [5, 10, 23, 28]. They are also related to the size of *crossing families* (pairwise crossing edges) determined by points in the plane [12, 40]. Perhaps a most notable question on quasi-planarity is a conjecture by Pach, Shahrokhi, and Szegedy [39] that for any fixed integer $k \geq 3$, there exists a constant c_k such that every n-vertex k-quasi-planar graph has at most $c_k n$ edges. This conjecture has been verified for $k = 3$ [5] and $k = 4$ [3].

A drawing of a graph is called *k-quasi-plane* if no k edges in the drawing pairwise cross, and a drawing is *quasi-plane* if it is 3-quasi-plane. For example, the drawing of a tree in Figure 1 is quasi-plane. This concept plays an important role in decompositions of geometric graphs: Aichholzer et al. [6] showed recently that the complete geometric graph on $2n$ points in the plane can always be decomposed into n quasi-plane spanning trees (but not necessarily into n plane spanning trees).

1.3 Our contributions

In Section 2 we present a randomized approximation algorithm with factor $O(\log n)$ for the MinPBST problem. Our algorithm computes a randomly shifted quadtree on the points, and then builds a planar bichromatic tree in a bottom-up fashion from the leaves of the quadtree towards the root. We then derandomize the algorithm by discretizing the random shifts. Our weight analysis shows that $|\text{MinBST}(S)| \leq |\text{MinPBST}(S)| \leq O(\log n) \cdot |\text{MinBST}(S)|$ for every set S of n bichromatic points, which implies that $\rho_n = O(\log n)$.

In Section 3.1 we prove that every MinBST is quasi-plane, i.e., no three edges pairwise cross in its inherited drawing (determined by the point set). In a sense, this means that MinBST is not far from plane graphs. In Section 3.2 we determine the maximum number of crossings in a MinBST. We conclude with a list of open problems in Section 4.

2 Approximation Algorithm for MinPBST

In this section we first present a randomized approximation algorithm for the MinPBST problem. Then we show how to derandomize the algorithm at the expense of increasing the running time by a quadratic factor. The following theorem summarizes our result in this section. Throughout this section we consider point sets in the plane that are in general position, that is, no three points lie on a straight line.

▶ **Theorem 1.** *There is a randomized algorithm that, given a set of n red and blue points in the plane in general position, returns a plane bichromatic spanning tree of expected weight at most $O(\log n)$ times the optimum, and runs in $O(n \log^2 n)$ time. The algorithm can be derandomized by increasing the running time by a factor of $O(n^2)$.*

Let S be a set of n red and blue points in the plane. To simplify our arguments we assume that n is a power of 2. Let OPT denote the length of a minimum bichromatic spanning tree on S (and note that OPT is an obvious lower bound for the length of a minimum *plane* bichromatic spanning tree on S). Our algorithm computes a plane bichromatic spanning tree of expected length $O(\log n) \cdot \mathrm{OPT}$.

2.1 Preliminaries for the algorithm

The following folklore lemma, though very simple, plays an important role in our construction.

▶ **Lemma 2.** *Every set of n red and blue points in general position in the plane, containing at least one red and at least one blue point, admits a plane bichromatic spanning tree. Such a tree can be computed in $O(n \log n)$ time.*

A proof of Lemma 2 can be found in [17]. Essentially such a tree can be constructed by connecting an arbitrary red point to all blue points (this partitions the plane into cones) and then connecting red points in each cone to a blue point on its boundary.

For a connected geometric graph G and a point q in the plane, we say that q *sees* an edge (a, b) of G if the interior of the triangle $\triangle qab$ is disjoint from vertices and edges of G. In other words, the entirity of the edge (a, b) is visible from q. The following lemma (that is implied from [33, Lemma 2.1]) also plays an important role in our construction.

▶ **Lemma 3.** *Let G be a connected plane geometric graph with n vertices and q be a point outside the convex hull of the vertices of G. Then q sees an edge of G. Such an edge can be found in $O(n \log n)$ time.*

Note that the condition that q lies outside of the convex hull of G is necessary, as otherwise q may not see any edge of G entirely. The application of this lemma to our algorithm is that if G is properly colored and a vertex sees an edge (a, b), then (q, a) or (q, b) is bichromatic and does not cross any edges of G. This idea was previously used in [31, 32, 33].

2.2 The algorithm

After a suitable scaling, we may assume that the smallest axis-aligned square containing S has side length 1. After a suitable translation, we may assume that the lower left corner of this square is the point $(1,1)$; and so its top right corner is $(2,2)$ as in Figure 2(a). Observe that

$$\mathrm{OPT} \geq 1.$$

Our algorithm employs a randomly shifted quadtree as in Arora's PTAS for the Euclidean TSP [13]. Let Q be a 2×2 axis-aligned square whose lower left corner is the origin – Q contains all points of S. Subdivide Q into four congruent squares, and recurse until Q is subdivided into squares of side length $1/n$ as in Figure 2(a). The depth of this recursion is $1 + \log n$. For the purpose of shifting, pick two real numbers x and y in the interval $[0,1]$ uniformly at random. Then translate Q such that its lower left corner becomes (x,y) as in Figure 2(b). This process is called a *random shift*. The points of S remain in Q after the shift. We obtain a quadtree subdivision of S of depth at most $1 + \log n$ with respect to the subdivision of Q, i.e., the lines of the quadtree subdivision are chosen from the lines of the subdivision of Q; see Figure 2(c). The resulting quadtree is called (randomly) *shifted quadtree*. We stop the recursive subdivision at squares that have size $1/n \times 1/n$ or that are *empty* (disjoint from S) or *monochromatic* (have points of only one color). Therefore a leaf-square of the quadtree may contain more than one point of S.

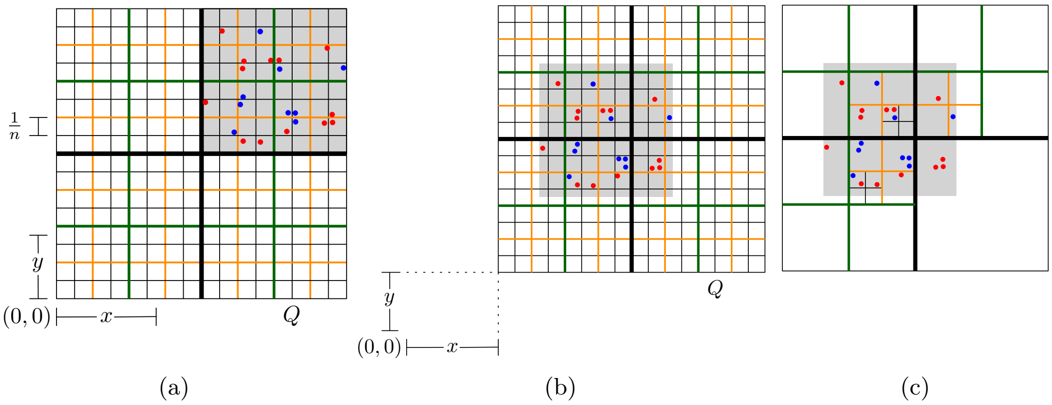


Figure 2 (a) Shaded square contains S. (b) Translated subdivision of Q. (c) Randomly shifted quadtree on points of S with respect to the subdivision of Q.

At the root level (which we may consider as level -1) we have Q which has size 2×2. At each recursive level $i = 0, 1, \ldots, \log n$ we have squares of size $1/2^i \times 1/2^i$ (Level 0 stands for the first time that we subdivide Q). Thus at level 0 we have four squares of size 1×1, and at level $\log n$ we have squares of size $1/n \times 1/n$. Our strategy is to use the shifted quadtree and compute an approximate solution in a bottom-up fashion from the leaves towards the root. For each square that is *bichromatic* (contains points of both colors) we will find a plane bichromatic spanning tree of its points. For monochromatic squares, we do not do anything. At the root level, we have Q which contains S and is bichromatic, so we will get an approximate plane bichromatic spanning tree of S.

At level $i = \log n$, we have squares of size $1/n \times 1/n$, and thus the length of any edge in such squares is at most $\sqrt{2}/n$. For each bichromatic square we compute a plane bichromatic tree arbitrarily, for instance by Lemma 2. For monochromatic squares we do nothing. The spanning trees for all the squares have less than n edges in total. Hence the total length of all these trees is at most

$$\frac{\sqrt{2}}{n} \cdot n = \sqrt{2} \leq \sqrt{2} \cdot \mathrm{OPT}.$$

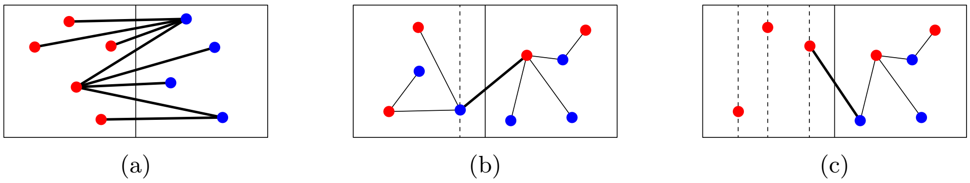


Figure 3 Merging two squares in the same row: (a) case 1, (b) case 2, and (c) case 3.

At each level $i < \log n$, we have already solved the problem within squares of level $i + 1$. Each square at level i has size $1/2^i \times 1/2^i$. If the square is monochromatic we do nothing. If the square is bichromatic, then it consists of four squares at level $i + 1$. We merge the solutions of the four squares to obtain a solution for the level i square as follows. First we merge the solutions in adjacent squares in the same row (we have two such pairs), and then merge the two solutions in the two rows. Thus each merge is performed on two solutions that are separated by a (vertical or horizontal) line. For the merge we apply one of the following cases:

(1) If each merge party is monochromatic but their union is bichromatic, then we construct an arbitrary plane bichromatic tree on the union, for example by Lemma 2. See Figure 3(a).

(2) If both merge parties are bichromatic (and hence are trees), then we take a point in one tree that is closest to the square (or rectangle) containing the other tree and merge them using Lemma 3. See Figure 3(b).

(3) If one part is bichromatic (a tree) and the other is monochromatic, then we first sort the points in the monochromatic square (or rectangle) in increasing order according to their distance to the bichromatic square (or rectangle). Then we merge the points (one at a time) with the current bichromatic tree by using Lemma 3. See Figure 3(c).

In each case, the merge produces a plane bichromatic tree in the level-i square. We process all squares in a bottom-up traversal of the quadtree. In the end, after processing level -1, we get a plane bichromatic spanning tree for points of S in square Q. Denote this tree by T.

2.3 Weight analysis

We start by introducing an alternative measurement for the length of the optimal tree, i.e., MinBST. This new measurement, denoted by OPT$'$, will be used to bound the length of our tree T. We say that a quadtree line is at level i if it contains a side of some level-i square. A side of a level-i square (that is not a leaf) gets subdivided to yield sides of two squares at level $i + 1$. Thus any quadtree line at level i is also at levels $i + 1, i + 2, \dots, \log n$.

For each level $i \in \{0, 1, \dots, \log n\}$ we define a parameter OPT'_i. Let T_{opt} be a MinBST for S. For each edge e of T_{opt}, define the indicator variable

$$X_i(e) = \begin{cases} 1 & \text{if } e \text{ intersects the boundary of a level-}i\text{ square of the quadtree,} \\ 0 & \text{otherwise.} \end{cases}$$

Now let

$$\mathrm{OPT}'_i = \sum_{e \in E(T_{\mathrm{opt}})} \frac{1}{2^i} X_i(e),$$

that is, a weighted sum of $X_i(e)$ over all edges of T_{opt}, and let

$$\mathrm{OPT}' = \sum_{i=0}^{\log n} \mathrm{OPT}'_i.$$

The new measure OPT' can be arbitrarily large compared to OPT in the worst case. For example if the optimal solution is a path consisting of $n-1$ edges of small length say $2/n$, such that each of them intersects the vertical line at level 0, then OPT is roughly 2 but OPT' is at least $n-1$. However, using the random shift at the beginning of our algorithm, we can show that the expected value of OPT' is not very large compared to OPT.

▶ **Lemma 4.** $\mathbb{E}[\mathrm{OPT}'] \leq \sqrt{2}(1 + \log n) \cdot \mathrm{OPT}$.

Proof. Consider any edge e of length ℓ in MinBST. Let ℓ_x denote the *x-span* of e, i.e., the difference of the x-coordinates of the two endpoints of e, and let ℓ_y be the *y-span* of e. Observe that $\ell_x + \ell_y \leq \sqrt{2}\ell$. The probability that e intersects a vertical line at any level i is

$$\Pr(e \text{ intersects a vertical line at level } i) \leq \ell_x 2^i,$$

because there are 2^i uniformly spaced but randomly shifted vertical lines at level i. Combining this with the probability of intersecting horizontal lines, the union bound yields

$$\Pr(X_i(e)) = \Pr(e \text{ intersects a line at level } i) \leq (\ell_x + \ell_y)2^i \leq \sqrt{2}\ell 2^i.$$

If e intersects a line at level i, then $1/2^i$ is added to OPT'_i, otherwise nothing is added. Thus the expected added value for e to OPT'_i is

$$\mathbb{E}\left[\frac{1}{2^i} X_i(e)\right] = \frac{1}{2^i} \cdot \Pr(X_i(e)) \leq \frac{1}{2^i} \cdot \sqrt{2}\ell 2^i = \sqrt{2}\ell.$$

Summation over all edges of the optimal tree T_{opt} gives $\mathbb{E}[\mathrm{OPT}'_i] \leq \sqrt{2} \cdot \mathrm{OPT}$, and summation over all levels yields $\mathbb{E}[\mathrm{OPT}'] \leq \sqrt{2}(1 + \log n) \cdot \mathrm{OPT}$. ◀

We already know that the total length of trees constructed in level $\log n$ is at most $\sqrt{2} \cdot \mathrm{OPT}$. Let E_i be the set of all edges that were added to T at level $i < \log n$ in the bottom-up construction. We establish a correspondence between the edges in E_i and the values added to OPT'. The edges of E_i were added by cases (1), (2), and (3). We consider each case separately.

(1) Assume that the union of merge parties has k points. Then we add $k-1$ edges of length at most $\sqrt{2} \cdot 1/2^i$ to E_i. For these points, MinBST also needs to have at least $k-1$ connections that intersect the boundaries of the squares involved in the merge, which have side length at least $1/2^{i+1}$. For each such edge we have added a value of $1/2^{i+1}$ to OPT'_{i+1}.

(2) To merge the two trees, we added just one edge of length at most $\sqrt{2} \cdot 1/2^i$ to E_i. For each merge party, MinBST needs at least one edge that crosses the boundary of its rectangle. These edges, however, need not be distinct (e.g., an edge can cross the boundary between the two rectangles). In any case, MinBST needs at least one edge that crosses the boundary of one of the two rectangles, for which we have added at least $1/2^{i+1}$ to OPT'_{i+1}.

(3) Assume that the monochromatic party has k points. Thus we added k edges of length at most $\sqrt{2} \cdot {}^1\!/_{2^i}$ to E_i. Again, MinBST needs at least k edges that cross the boundary of the squares involved in the merge; and for each such edge we have added at least ${}^1\!/_{2^{i+1}}$ to OPT'_{i+1}.

Therefore the total length of all edges that were added to T at level i is at most $|E_i| \cdot {}^{\sqrt{2}}\!/_{2^i}$. Analogously, the total value that has been added to OPT'_{i+1} is at least ${}^1\!/_2 \cdot |E_i| \cdot {}^1\!/_{2^{i+1}} = |E_i| \cdot {}^1\!/_{2^{i+2}}$. The multiplicative factor ${}^1\!/_2$ comes from the fact that the two endpoints of an edge of the optimal tree could be involved in two separate merge operations. By summing the length of edges added to T in all levels and considering Lemma 4 we get

$$\begin{aligned}\mathbb{E}[|T|] &\le \sqrt{2} \cdot \mathrm{OPT} + \mathbb{E}\left[\sum_{i=0}^{\log n-1} |E_i| \frac{\sqrt{2}}{2^i}\right] \le \sqrt{2} \cdot \mathrm{OPT} + 4\sqrt{2} \cdot \mathbb{E}\left[\sum_{i=0}^{\log n-1} |E_i| \frac{1}{2^{i+2}}\right] \\ &\le \sqrt{2} \cdot \mathrm{OPT} + 4\sqrt{2} \cdot \mathbb{E}\left[\sum_{i=0}^{\log n-1} \mathrm{OPT}'_{i+1}\right] \le \sqrt{2} \cdot \mathrm{OPT} + 4\sqrt{2} \cdot \mathbb{E}[\mathrm{OPT}'] \\ &\le \sqrt{2} \cdot \mathrm{OPT} + 4\sqrt{2} \cdot \sqrt{2}(1+\log n) \cdot \mathrm{OPT} = O(\log n) \cdot \mathrm{OPT}.\end{aligned}$$

2.4 Derandomization

In the algorithm of Section 2.2, we shifted the 2×2 square Q by a real vector (x, y), where x and y are chosen independently and uniformly at random from the interval $[0, 1]$. We now *discretize* the random shift, and choose x and y independently and uniformly at random from the finite set $\{0, {}^1\!/_n, {}^2\!/_n, \ldots, {}^{n-1}\!/_n\}$. We call this process the *discrete random shift.* We show that the proof of Lemma 4 can be adapted under this random experiment with a larger constant coefficient. Therefore we can derandomize the algorithm by trying all random choices of (x, y) for the shift and return the shortest tree over all choices. This increases the running time by a factor of $O(n^2)$.

▶ **Lemma 5.** *Under the discrete random shift, we have* $\mathbb{E}[\mathrm{OPT}'] \le (\sqrt{2}+2)(1+\log n) \cdot \mathrm{OPT}$.

Proof. Consider any edge $e = ab$ of length ℓ in the optimal tree T_{opt}. The two endpoints of e are points $a = (a_x, a_y)$ and $b = (b_x, b_y)$. Denote the orthogonal projection of e to the x- and y-axes by e_x and e_y, respectively, and observe that $e_x = [\min\{a_x, b_x\}, \max\{a_x, b_x\}]$ and $e_y = [\min\{a_y, b_y\}, \max\{a_y, b_y\}]$. We discretize these intervals as follows. Replace each endpoint of the interval e_x with the closest rationals of the form ${}^k\!/_n + {}^1\!/_{2n}$, and let e'_x be the resulting interval; similarly we obtain e'_y from e_y. Observe that e_x intersects a vertical line of the form $x = {}^a\!/_n$, $a \in \mathbb{Z}$, if and only if e'_x does. Let ℓ'_x and ℓ'_y denote the lengths of intervals e'_x and e'_y, respectively. By construction, we have $\ell'_x \le \ell_x + {}^1\!/_n$ and $\ell'_y \le \ell_y + {}^1\!/_n$. Consequently, the probability that e intersects a vertical line at any level i is

$$\begin{aligned}\Pr(e \text{ intersects a vertical line at level } i) &= \Pr(e'_x \text{ intersects a vertical line at level } i) \\ &\le \ell'_x 2^i \le \left(\ell_x + {}^1\!/_n\right) 2^i,\end{aligned}$$

because there are 2^i uniformly spaced but randomly shifted vertical lines at level i. Combining this with the probability of intersecting horizontal lines, the union bound yields

$$\Pr(X_i(e)) = \Pr(e \text{ intersects a line at level } i) \le \left(\ell_x + \ell_y + {}^2\!/_n\right) 2^i \le \left(\sqrt{2}\ell + {}^2\!/_n\right) 2^i.$$

The expected added value for e to OPT'_i is

$$\mathbb{E}\left[\frac{1}{2^i}X_i(e)\right] = \frac{1}{2^i}\cdot \Pr(X_i(e)) \le \frac{1}{2^i}\cdot\left(\sqrt{2}\ell + \frac{2}{n}\right)2^i = \sqrt{2}\ell + \frac{2}{n}.$$

Summation over all edges of the optimal tree T_{opt} gives $\mathbb{E}[\mathrm{OPT}'_i] \le \sqrt{2}\cdot \mathrm{OPT} + 2 \le (\sqrt{2}+2)\cdot\mathrm{OPT}$, and summation over all levels yields $\mathbb{E}[\mathrm{OPT}'] \le (\sqrt{2}+2)(1+\log n)\cdot\mathrm{OPT}$. ◀

The initial shifted quadtree has depth $O(\log n)$ and has $O(n)$ leaves. Thus it can be computed in $O(n\log n)$ time by a divide-and-conquer sorting-based algorithm [26]. From a result of [17] (cf. Lemma 2) it follows that the bichromatic trees at the leaves of the quadtree can be computed in total $O(n\log n)$ time. From a result of [30] and [33] (cf. Lemma 3) it follows that the total merge time in each level of the quadtree is $O(n\log n)$. Summing over all levels, the running time of our algorithm is $O(n\log^2 n)$.

2.5 Generalization to more colors

Our approximation algorithm for the MinPBST (minimum plane two-colored spanning tree) can be generalized to more colors. In this general setting, we are given a colorful point set S and we want to find a spanning tree on S with properly colored edges, i.e., the two endpoints of every edge should be of different colors. The same quadtree approach would give a plane properly-colored spanning tree. The analysis and the approximation ratio would be the same mainly because whenever we introduce some edges to merge two squares, the optimal solution must have the same number of edges that cross the boundaries of the squares. Also the running time remains the same because Lemma 2 and Lemma 3 carry over to multicolored point sets.

3 Crossing Patterns in MinBST

In this section, we prove that MinBST is quasi-plane for every 2-colored point set in general position (Section 3.1), and then use this result to determine the maximum number of crossings in MinBST for a set of n bichromatic points in the plane (Section 3.2).

3.1 Quasi-planarity

Let S be a set of red and blue points in the plane. To differentiate between the points we denote the red points by $r_1, r_2, \dots$ and the blue points by $b_1, b_2, \dots$. Let T be a MinBST for S. For two distinct edges e_1 and e_2 of T we denote the unique shortest path between e_1 and e_2 in T by $\delta(e_1, e_2)$. This path contains exactly one endpoint of e_1 and one endpoint of e_2.

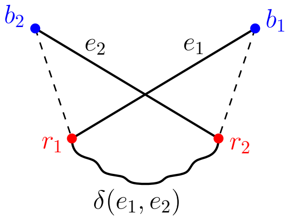


Figure 4 Illustration of the proof of Lemma 6. Uncrossing a pair of crossing edges.

▶ **Lemma 6.** *Let e_1 and e_2 be two edges of T that cross each other. Then the endpoints of $\delta(e_1, e_2)$ have different colors.*

Proof. Let $e_1 = (r_1, b_1)$ and $e_2 = (r_2, b_2)$. Suppose, for the sake of contradiction, that the endpoints of $\delta(e_1, e_2)$ are of the same color, w.l.o.g. red. Then the endpoints of $\delta(e_1, e_2)$ are r_1 and r_2 as in Figure 4. In this case, we can replace edges (r_1, b_1) and (r_2, b_2) of T by two new edges (r_1, b_2) and (r_2, b_1) and obtain a new bichromatic spanning tree T'. By the triangle inequality (applied to each of the two triangles induced by the crossing), the total length of the two new edges is smaller than the total length of the two orginal edges. Hence T' is shorter than T, contradicting the minimality of T. ◀

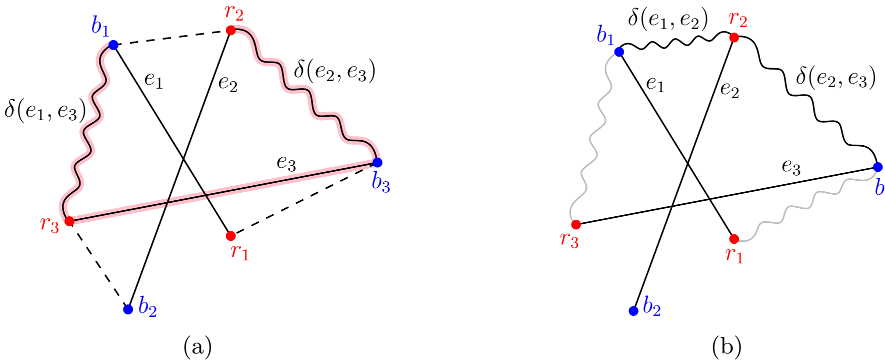


Figure 5 (a) Replacing e_1, e_2, e_3 by $(r_1, b_3), (r_3, b_2), (r_2, b_1)$; the highlighted path is $\delta(e_1, e_2)$. (b) Getting a cycle in the union of $\delta(e_1, e_2)$, $\delta(e_2, e_3)$, $\delta(e_1, e_3)$, together with e_1 or e_3. Gray paths represent the two possible choices for $\delta(e_1, e_3)$.

▶ **Theorem 7.** *Every Euclidean minimum bichromatic spanning tree is quasi-plane.*

Proof. Let T be a Euclidean minimum bichromatic spanning tree. We prove that no three edges of T can pairwise cross each other. This will imply that T is quasi-plane. The proof proceeds by contradiction. Suppose that three edges of T, say $e_1 = (r_1, b_1)$, $e_2 = (r_2, b_2)$ and $e_3 = (r_3, b_3)$, pairwise cross each other as in Figure 5. We consider the following two cases:

1. $\delta(e_i, e_j)$ contains e_k for some permutation of the indices with $\{i, j, k\} = \{1, 2, 3\}$.
   After a suitable relabeling assume that $\delta(e_1, e_2)$ contains e_3. Then $\delta(e_1, e_3)$ and $\delta(e_2, e_3)$ are sub-paths of $\delta(e_1, e_2)$ and they do not contain e_2 and e_1 respectively. Assume without loss of generality (and by Lemma 6) that the endpoints of $\delta(e_1, e_2)$ are r_2 and b_1 as in Figure 5(a); $\delta(e_1, e_2)$ is highlighted in the figure. For the rest of our argument we will use a result of [20] that for an even number of (monochromatic) points in the plane, a perfect matching with pairwise crossing edges is the unique maximum-weight matching (without considering the colors). This means that $M = \{e_1, e_2, e_3\}$ is the maximum matching for the point set $\{r_1, r_2, r_3, b_1, b_2, b_3\}$. Therefore M is longer than $R = \{(r_1, b_3), (r_2, b_1), (r_3, b_2)\}$, which is a (bichromatic) matching for the same points. By replacing the edges of M in T with the edges of R, we obtain a shorter tree T', contradicting the minimality of T. To verify that T' is a tree imagine replacing the edges one at a time. If we add (r_1, b_3) first, then we create a cycle that contains e_1, and thus by removing e_1 we obtain a valid tree. Similarly we can replace e_2 by (r_3, b_2) and e_3 by (r_2, b_1).
2. $\delta(e_i, e_j)$ does not contain e_k for any permutation of the indices with $\{i, j, k\} = \{1, 2, 3\}$.
   Consider the path $\delta(e_1, e_2)$ and assume without loss of generality (and by Lemma 6) that its endpoints are r_2 and b_1 as in Figure 5(b). Now consider $\delta(e_2, e_3)$. This path cannot

have r_3 and b_2 as its endpoints because otherwise the path $\delta(e_1, e_3)$ would contain e_2, contradicting the assumption of the current case. Therefore the endpoints of $\delta(e_2, e_3)$ are r_2 and b_3. Now consider the path $\delta(e_1, e_3)$. If its endpoints are r_1 and b_3, then the union of $\delta(e_1, e_2)$, $\delta(e_2, e_3)$, and $\delta(e_1, e_3)$ contains a path between r_1 and b_1 that does not go through e_1; the union of this path and e_1 is a cycle in T. Similarly, if the endpoints of $\delta(e_1, e_3)$ are r_3 and b_1, then the union of $\delta(e_1, e_2)$, $\delta(e_2, e_3)$, $\delta(e_1, e_3)$, and e_3 contains a cycle. Both cases lead to a contradiction as T has no cycle. ◀

3.2 Maximum number of crossings

Given that a MinBST is quasi-plane (Theorem 7), one wonders how many crossings it can have. As illustrated in Figure 1, the number of crossings per edge can be linear in the number of points, and the total number of crossings can be quadratic. We give tight upper bounds for both quantities (Propositions 9–10), and also show that MinBST always has a crossing-free edge (Proposition 8).

▶ **Proposition 8.** *For every finite set of bichromatic points in the plane in general position, every MinBST contains a closest bichromatic pair as an edge. Moreover, no such edge is intersected by other edges of the MinBST.*

Proof. We prove both parts by contradiction. For the first part assume that the MinBST does not contain an edge between any closest bichromatic pair. Let $\{r_1, b_1\}$ be a closest bichromatic pair. By adding (r_1, b_1) to the tree we obtain a cycle in which (r_1, b_1) is the shortest edge. By removing any other edge from the cycle we obtain a bichromatic spanning tree of a shorter length. This contradicts the minimality of the original MinBST.

For the second part let $\{r_1, b_1\}$ be a closest bichromatic pair that appears as an edge in the MinBST. Hence $e_1 = (r_1, b_1)$ crosses some edge $e_2 = (r_2, b_2)$. Without loss of generality (and by Lemma 6), assume that the endpoints of $\delta(e_1, e_2)$ are r_1 and b_2. If $|r_2b_1| < |r_2b_2|$, then by replacing (r_2, b_2) with (r_2, b_1) we obtain a shorter bichromatic spanning tree, a contradiction. Assume that $|r_2b_1| \geq |r_2b_2|$. Since $\{r_1, b_1\}$ is a closest bichromatic pair we have $|r_1b_2| \geq |r_1b_1|$. Adding the two inequalities yields $|r_1b_2| + |r_2b_1| \geq |r_1b_1| + |r_2b_2|$. Note, however, that the vertices of any two crossing edges form a convex quadrilateral. By the triangle inequality, the total length of the two diagonals of a convex quadrilateral is strictly more than the total length of any pair of opposite edges, yielding $|r_1b_2|+|r_2b_1| < |r_1b_1|+|r_2b_2|$, a contradiction. ◀

▶ **Proposition 9.** *For every set of $n \geq 2$ bichromatic points in the plane in general position, every MinBST has at most $\lfloor n^2/4 \rfloor - n + 1$ crossings, and this bound is the best possible.*

Proof. To verify that the claimed bound can be attained, consider the construction in Figure 1 where the two top clusters have $\lfloor n/2 \rfloor - 1$ and $\lceil n/2 \rceil - 1$ points. Then the total number of crossings in MinBST(S) is $(\lfloor n/2 \rfloor - 1) \cdot (\lceil n/2 \rceil - 1)$, which is equal to $\lfloor n^2/4 \rfloor - n + 1$ because n is an integer.

For an upper bound, let S be a set of n bichromatic points in general position. Define the *crossing graph* G_{cr} of MinBST(S), where the vertices of G_{cr} correspond to the edges of MinBST(S) and edges of G_{cr} represent crossings between the edges of MinBST(S). Note that G_{cr} has $n-1$ vertices where one of them is of degree 0 by Proposition 8. By Theorem 7, G_{cr} is triangle-free. Therefore, by Turán's theorem [7], G_{cr} has at most $(\lfloor n/2 \rfloor - 1) \cdot (\lceil n/2 \rceil - 1)$ edges. Consequently, MinBST(S) has at most this many crossings. ◀

▶ **Proposition 10.** *For every set of $n \geq 3$ bichromatic points in the plane in general position, every edge of a MinBST crosses at most $n-3$ other edges, and this bound is the best possible.*

Proof. To verify that the claimed bound can be attained, consider the construction in Figure 1 and replace one of the top clusters by a single point and the other by $n-3$ points. Then all points in this cluster have degree 1 in MinBST(S), these $n-3$ leaves all cross one edge of MinBST.

For the upper bound, notice that a MinBST has $n-1$ edges, one of which is crossing-free by Proposition 8. Consequently, an edge in a MinBST can cross at most $n-3$ other edges. ◀

4 Conclusions and Open Problems

We conclude with a collection of open problems raised by our results. We have presented a $O(\log n)$-approximation algorithm for the MinPBST problem for a set of n bichromatic points in the plane, and showed that $\rho_n \leq O(\log n)$. Recall that the current best lower bound is $\rho_n \geq 3/2$ for all $n \geq 4$ [29]. It remains open whether a constant-factor approximation is possible, whether the problem is APX-hard, and whether ρ_n is bounded by a constant.

It is also natural to investigate whether there is an (approximation) algorithm that, given a bichromatic point set and an integer d, finds a *minimum plane bichromatic tree of maximum degree at most d* (or reports that none exists). It is known that any set of n red and n blue points in general position admits a plane bichromatic spanning tree of maximum degree at most three [34]; but there are n red and n blue points in convex position that do not admit a bichromatic plane spanning path [8]. For the general case of n red and m blue points, with $n \geq m$, there exists a plane bichromatic spanning tree of maximum degree at most $\max\{3, \lceil {}^{n-1}/_{m} \rceil + 1\}$ and this is the best upper bound [19].

We have shown that MinBST is quasi-plane, which means that the crossing graph G_{cr} of MinBST is triangle-free. Figure 1 shows that G_{cr} can have 4-cycles (and even cycles of any lengths). Can the crossing graph G_{cr} of MinBST contain an odd cycle (e.g., a 5-cycle)? Can every MinBST be decomposed into a constant number of planar straight-line graphs?

References

1 Manuel Abellanas, Jesus García-Lopez, Gregorio Hernández-Peñalver, Marc Noy, and Pedro A. Ramos. Bipartite embeddings of trees in the plane. *Discret. Appl. Math.*, 93(2-3):141–148, 1999. doi:10.1016/S0166-218X(99)00042-6.

2 A. Karim Abu-Affash, Sujoy Bhore, Paz Carmi, and Joseph S. B. Mitchell. Planar bichromatic bottleneck spanning trees. *J. Comput. Geom.*, 12(1):109–127, 2021. doi:10.20382/JOCG.V12I1A5.

3 Eyal Ackerman. On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discret. Comput. Geom.*, 41(3):365–375, 2009. doi:10.1007/S00454-009-9143-9.

4 Pankaj K. Agarwal. Partitioning arrangements of lines I: An efficient deterministic algorithm. *Discret. Comput. Geom.*, 5:449–483, 1990. doi:10.1007/BF02187805.

5 Pankaj K. Agarwal, Boris Aronov, János Pach, Richard Pollack, and Micha Sharir. Quasi-planar graphs have a linear number of edges. *Combinatorica*, 17(1):1–9, 1997. doi:10.1007/BF01196127.

6 Oswin Aichholzer, Johannes Obenaus, Joachim Orthaber, Rosna Paul, Patrick Schnider, Raphael Steiner, Tim Taubner, and Birgit Vogtenhuber. Edge partitions of complete geometric graphs. In *Proc. 38th Symposium on Computational Geometry (SoCG)*, volume 224 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl, 2022. doi:10.4230/LIPICS.SOCG.2022.6.

7 Martin Aigner and Günter M. Ziegler. Turán's graph theorem. In *Proofs from THE BOOK*, chapter 41, pages 285–289. Springer-Verlag, 6th edition, 2018. doi:10.1007/978-3-662-57265-8_41.

8 Jin Akiyama and Jorge Urrutia. Simple alternating path problem. *Discret. Math.*, 84(1):101–103, 1990. doi:10.1016/0012-365X(90)90276-N.

9 Carlos Alegría, David Orden, Carlos Seara, and Jorge Urrutia. Separating bichromatic point sets in the plane by restricted orientation convex hulls. *J. Glob. Optim.*, 85(4):1003–1036, 2023. doi:10.1007/S10898-022-01238-9.

10 Patrizio Angelini, Michael A. Bekos, Franz J. Brandenburg, Giordano Da Lozzo, Giuseppe Di Battista, Walter Didimo, Michael Hoffmann, Giuseppe Liotta, Fabrizio Montecchiani, Ignaz Rutter, and Csaba D. Tóth. Simple k-planar graphs are simple $(k+1)$-quasiplanar. *J. Comb. Theory B*, 142:1–35, 2020. doi:10.1016/J.JCTB.2019.08.006.

11 Bogdan Armaselu and Ovidiu Daescu. Dynamic minimum bichromatic separating circle. *Theor. Comput. Sci.*, 774:133–142, 2019. doi:10.1016/J.TCS.2016.11.036.

12 Boris Aronov, Paul Erdős, Wayne Goddard, Daniel J. Kleitman, Michael Klugerman, János Pach, and Leonard J. Schulman. Crossing families. *Combinatorica*, 14(2):127–134, 1994. doi:10.1007/BF01215345.

13 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

14 Sanjeev Arora and Kevin L. Chang. Approximation schemes for degree-restricted MST and red-blue separation problems. *Algorithmica*, 40(3):189–210, 2004. doi:10.1007/s00453-004-1103-4.

15 Sayan Bandyapadhyay, Aritra Banik, Sujoy Bhore, and Martin Nöllenburg. Geometric planar networks on bichromatic collinear points. *Theor. Comput. Sci.*, 895:124–136, 2021. doi:10.1016/J.TCS.2021.09.035.

16 Sergei Bespamyatnikh, David G. Kirkpatrick, and Jack Snoeyink. Generalizing ham sandwich cuts to equitable subdivisions. *Discret. Comput. Geom.*, 24(4):605–622, 2000. doi:10.1007/s004540010065.

17 Ahmad Biniaz, Prosenjit Bose, Kimberly Crosbie, Jean-Lou De Carufel, David Eppstein, Anil Maheshwari, and Michiel H. M. Smid. Maximum plane trees in multipartite geometric graphs. *Algorithmica*, 81(4):1512–1534, 2019. doi:10.1007/S00453-018-0482-X.

18 Ahmad Biniaz, Prosenjit Bose, David Eppstein, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Spanning trees in multipartite geometric graphs. *Algorithmica*, 80(11):3177–3191, 2018. doi:10.1007/S00453-017-0375-4.

19 Ahmad Biniaz, Prosenjit Bose, Anil Maheshwari, and Michiel H. M. Smid. Plane bichromatic trees of low degree. *Discret. Comput. Geom.*, 59(4):864–885, 2018. doi:10.1007/S00454-017-9881-Z.

20 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Euclidean maximum matchings in the plane—local to global. In *Proceedings of the 17th International Symposium on Algorithms and Data Structures (WADS)*, volume 12808 of *LNCS*, pages 186–199. Springer, 2021. doi:10.1007/978-3-030-83508-8_14.

21 Magdalene G. Borgelt, Marc J. van Kreveld, Maarten Löffler, Jun Luo, Damian Merrick, Rodrigo I. Silveira, and Mostafa Vahedi. Planar bichromatic minimum spanning trees. *J. Discrete Algorithms*, 7(4):469–478, 2009. doi:10.1016/J.JDA.2008.08.001.

22 Otakar Borůvka. O jistém problému minimálním. *Praca Moravske Prirodovedecke Spolecnosti*, 3(3):37–58, 1926.

23 Vasilis Capoyleas and János Pach. A Turán-type theorem on chords of a convex polygon. *J. Comb. Theory B*, 56(1):9–15, 1992. doi:10.1016/0095-8956(92)90003-G.

24 Timothy M. Chan. On the bichromatic k-set problem. *ACM Trans. Algorithms*, 6(4):62:1–62:20, 2010. doi:10.1145/1824777.1824782.

25 Timothy M. Chan and Bryan T. Wilkinson. Bichromatic line segment intersection counting in $O(n\sqrt{\log n})$ time. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG)*, Toronto, ON, 2011. URL: https://cccg.ca/proceedings/2011/papers/paper83.pdf.

26 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Quadtrees. In *Computational Geometry: Algorithms and Applications*, chapter 14, pages 307–322. Springer, Berlin, 2008. doi:10.1007/978-3-540-77974-2_14.

27 Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Henk Meijer, Mark H. Overmars, and Sue Whitesides. Separating point sets in polygonal environments. *Int. J. Comp. Geom. Appl.*, 15(4):403–420, 2005. doi:10.1142/S0218195905001762.

28 Jacob Fox, János Pach, and Andrew Suk. Quasiplanar graphs, string graphs, and the Erdős-Gallai problem. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 219–231, 2022. doi:10.1007/978-3-031-22203-0_16.

29 Magdalene Grantson, Henk Meijer, and David Rappaport. Bi-chromatic minimum spanning trees. In *Proceedings of the 21st European Workshop on Computational Geometry (EwCG)*, pages 199–202, Eindhoven, 2005. URL: https://www.win.tue.nl/EWCG2005/Proceedings/51.pdf.

30 John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32(2):249–267, 1992. doi:10.1007/BF01994880.

31 Michael Hoffmann, Bettina Speckmann, and Csaba D. Tóth. Pointed binary encompassing trees: Simple and optimal. *Comput. Geom.*, 43(1):35–41, 2010. doi:10.1016/j.comgeo.2006.12.005.

32 Michael Hoffmann and Csaba D. Tóth. Vertex-colored encompassing graphs. *Graphs and Combinatorics*, 30(4):933–947, 2014. doi:10.1007/s00373-013-1320-1.

33 Ferran Hurtado, Mikio Kano, David Rappaport, and Csaba D. Tóth. Encompassing colored planar straight line graphs. *Comput. Geom.*, 39(1):14–23, 2008. doi:10.1016/J.COMGEO.2007.05.006.

34 Atsushi Kaneko. On the maximum degree of bipartite embeddings of trees in the plane. In *Discrete and Computational Geometry (JCDCG)*, volume 1763 of *LNCS*, pages 166–171, Heidelberg, 1998. Springer. doi:10.1007/978-3-540-46515-7_13.

35 Atsushi Kaneko and Mikio Kano. Discrete geometry on red and blue points in the plane — a survey. In Boris Aronov, Saugata Basu, János Pach, and Micha Sharir, editors, *Discrete and Computational Geometry*, volume 25 of *Algorithms and Combinatorics*, pages 551–570. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-642-55566-4_25.

36 Mikio Kano, Kazuhiro Suzuki, and Miyuki Uno. Properly colored geometric matchings and 3-trees without crossings on multicolored points in the plane. In *Discrete and Computational Geometry and Graphs (JCDCGG)*, LNCS, pages 96–111, Cham, 2013. Springer. doi:10.1007/978-3-319-13287-7_9.

37 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. doi:10.1090/S0002-9939-1956-0078686-7.

38 Harry G. Mairson and Jorge Stolfi. Reporting and counting intersections between two sets of line segments. In Rae A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume 40 of *NATO ASI Series*, pages 307–325. Springer, Heidelberg, 1988. doi:10.1007/978-3-642-83539-1_11.

39 János Pach, Farhad Shahrokhi, and Mario Szegedy. Applications of the crossing number. *Algorithmica*, 16(1):111–117, 1996. doi:10.1007/BF02086610.

40 János Pach, Natan Rubin, and Gábor Tardos. Planar point sets determine many pairwise crossing segments. *Advances in Mathematics*, 386:107779, 2021. doi:10.1016/j.aim.2021.107779.

41 Robert C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36, 1957.

Constrained Two-Line Center Problems

Taehoon Ahn
Graduate School of Artificial Intelligence,
Pohang University of Science and Technology, Republic of Korea

Sang Won Bae
Division of Artificial Intelligence and Computer Science,
Kyonggi University, Suwon, Republic of Korea

Abstract

Given a set P of n points in the plane, the two-line center problem asks to find two lines that minimize the maximum distance from each point in P to its closer one of the two resulting lines. The currently best algorithm for the problem takes $O(n^2 \log^2 n)$ time by Jaromczyk and Kowaluk in 1995. In this paper, we present faster algorithms for three variants of the two-line center problem in which the orientations of the resulting lines are constrained. Specifically, our algorithms solve the problem in $O(n \log n)$ time when the orientations of both lines are fixed; in $O(n \log^3 n)$ time when the orientation of one line is fixed; and in $O(n^2 \alpha(n) \log n)$ time when the angle between the two lines is fixed, where $\alpha(n)$ denotes the inverse Ackermann function.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases two-line center problem, geometric location problem, geometric optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.5

Related Version *Full Version*: `https://arxiv.org/abs/2409.13304` [7]

Funding *Taehoon Ahn*: Supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2019-0-01906).
Sang Won Bae: Supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00251168).

1 Introduction

Given a set P of n points in the plane $\mathbb{R}^2$, the *two-line center problem* asks to find two lines that minimize the maximum distance from each point in P to its closer one of the two resulting lines. In 1991, Agarwal and Sharir [4] presented the first subcubic $O(n^2 \log^5 n)$-time algorithm for the two-line center problem, in which they solved the decision version in $O(n^2 \log^3 n)$ time using their machinery [3] to maintain the width of a point set under a prescribed sequence of changes and then to apply the parametric search technique. (See also its full version [5].) In 1995, Jaromczyk and Kowaluk [24] presented an $O(n^2 \log^2 n)$-time algorithm and also discussed an $O(n^2 \log n)$-time decision algorithm. Glozman et al. [21, 22] exhibited how any D-time decision algorithm for the two-line center problem can be converted to an optimization algorithm of $O(n^2 \log n + D \log n)$ time using sorted matrices. Later, Katz and Sharir [26] introduced an expander-based approach and showed how to solve the problem in $O(n^2 \log^3 n + D \log n)$ time. There was no significant progress since then and $O(n^2 \log^2 n)$ still remains the best known upper bound [22, 24].

This paper addresses constrained variants of the two-line center problem, and aims to provide efficient algorithms for the constrained problems, particularly faster than $O(n^2 \log^2 n)$ time, and to provide new observations and algorithmic techniques for any future breakthrough on the problem. The currently fastest algorithm by Jaromczyk and Kowaluk [24] indeed considers several constrained problems, tackled by different methods. Though not having explicitly mentioned in [24], their approach yields an $O(n \log^2 n)$-time algorithm when a fixed point in P should be the farthest to the resulting lines, after an $O(n^2)$-time preprocessing. Recently, Bae [10] presented an $O(n^2)$-time algorithm for the two-*parallel*-line center problem, in which the two resulting lines are supposed to be parallel.

In this paper, we solve three variants of the two-line center problem, constrained about the orientations of the resulting two lines. Following summarizes our results and approaches:

(1) *(Two fixed orientations)* Given two orientations θ and ϕ, we present an $O(n \log n)$-time algorithm that solves the two-line center problem in which the two resulting lines are constrained to have orientations θ and ϕ. If the input points P are given as a sorted list in one of the specified orientations, then the running time can be reduced to $O(n)$.

(2) *(One fixed orientation)* Given an orientation ϕ, we present an $O(n \log^3 n)$-time algorithm that solves the two-line center problem in which one of the resulting lines is constrained to have orientation ϕ. We first devise an $O(n \log^2 n)$-time decision algorithm for this constrained problem using the data structure by Agarwal and Sharir [3]. In spite of having such an efficient decision algorithm, it is not immediate to achieve a sub-quadratic time optimization algorithm by applying known techniques; as introduced above, all known techniques for the two-line center problem require at least quadratic-time additional overhead [5, 22, 26]. To overcome this difficulty, we use our decision algorithm as a subroutine to find an interval narrow enough to reduce the possible number of candidate configurations to $O(n)$ and apply the dynamic width structure by Chan [13].

(3) *(Fixed angle of intersection)* Given a real β, we present an $O(n^2\alpha(n) \log n)$-time algorithm that solves the two-line center problem in which the two resulting lines are constrained to make angle β, where $\alpha(n)$ denotes the inverse Ackermann function. As in the second problem, we start by presenting a decision algorithm and apply the known technique [22] to obtain a favorably narrow interval that contains the optimal width value. We then consider a sweeping process in which we rotate a strip of variable width within the interval, and prove that if suffices to find an optimal solution by simulating the process.

To our best knowledge, the three constrained problems have not been considered in the literature. Note that the two-parallel-line center problem studied in [10] is a more constrained variant of our problems: In the first problem (of two fixed orientations), the special case of $\theta = \phi$ can be solved in $O(n)$ time, and the third problem (of fixed angle) for $\beta = 0$ indeed asks to find a two-parallel-line center, which can be solved in $O(n^2)$ time [10].

Due to space limit, most proofs are omitted, but can be found in the full version [7].

Related work

The two-line center problem is a special case of the *k-line center* problem for $k \geqslant 1$. For $k = 1$, known as the *width* problem, one can solve the problem in $O(n \log n)$ time [29], or in $O(n)$ time if the convex hull of P is given [31]. In three dimensions, the width of n points in $\mathbb{R}^3$ can be computed in $O(n^{3/2+\epsilon})$ expected time by Agarwal and Sharir [6]. In higher dimensions $d \geqslant 4$, Chan [12] showed how to compute the width in $O(n^{\lceil d/2 \rceil})$ time. In the plane $\mathbb{R}^2$, the k-line center problem is known to be NP-hard when k is part of the input [28], while efficient approximation algorithms are known [1, 2]. Agarwal et al. [2] presented an efficient approximation algorithm. Exact algorithms for $k \leqslant 2$ are presented as aforementioned, while any nontrivial exact algorithm for $k \geqslant 3$ is, however, unknown. An efficient $(1 + \epsilon)$-approximation algorithm for $k = 2$ is presented by Agarwal et al. [1]. Very recently, several constrained variants of the k-line center problem and its generalization in high dimensions have been considered. Das et al. [16] presented an approximation algorithm for the k-line center problem where the resulting lines are constrained to be axis-parallel. Chung et al. [15] considered a variant of the parallel k-line center problem. Ahn et al. [8] presented first algorithms for the problem of finding two parallel *slabs* in $\mathbb{R}^d$ for $d \geqslant 3$.

Not being restricted to the line center problems, there have been an enormous amount of results on constrained variants of those problems of finding optimal locations of one or more geometric shapes enclosing input objects. Such results on constrained problems usually provided more efficient solutions than those for the original (unconstrained) problems or played important roles as stepping stones to later breakthroughs. Constrained two-square problems [25] and the problem of covering points by two disjoint rectangles [27] are such examples.

Some preliminaries

A *strip* σ is the closed region between two parallel lines and its *width* is the distance between the two lines. A pair of two strips will be called a *two-strip* and the width of a two-strip mean the larger width of its two members. Note that the two-line center problem is equivalent to the problem of finding a two-strip of minimum width that enclose given points. The *orientation* of a line is a real value $\theta \in [0, \pi)$ such that θ is the angle swept from a horizontal line in counterclockwise direction to the line. Similarly, a strip is said to have an orientation θ when its bounding lines are in orientation θ. For any set P of points and orientation $\theta \in [0, \pi)$, we denote by $\sigma_\theta(P)$ the minimum-width strip in orientation θ that encloses P. We denote $\text{width}_\theta(P) := \text{width}(\sigma_\theta(P))$. The *width* of point set P, denoted by $\text{width}(P)$, is the smallest width of a strip that encloses P.

2 Two fixed orientations

In this section, we consider the first constrained problem where the orientations of two line centers should given values θ and ϕ. We assume that $\phi = 0$ without loss of generality.

We start by sorting the n points in P in the nondecreasing order of y-coordinates and let $p_1, \ldots, p_n \in P$ be in this order. For $0 \leqslant i \leqslant n$, let $P_i := \{p_1, \ldots, p_i\}$ and $\overline{P}_i := P \setminus P_i = \{p_{i+1}, \ldots, p_n\}$. It is straightforward in $O(n)$ time to incrementally construct the strips $\sigma_\theta(P_i)$ and $\sigma_\theta(\overline{P}_i)$ in orientation θ for all $0 \leqslant i \leqslant n$. We then observe the following.

▶ **Lemma 1.** *Given P as a sorted list as above, P can be processed in $O(n)$-time so that $\sigma_\theta(P_i \cup \overline{P}_j)$ can be answered in $O(1)$ time for any query pair (i, j) of indices.*

Consider any minimum-width two-strip (σ_1, σ_2) enclosing P such that its orientations are 0 and θ, respectively. Observe that σ_1 includes a contiguous sequence $p_{i+1}, \ldots, p_j$ of points in P for some indices $0 \leqslant i \leqslant j \leqslant n$, while σ_2 covers the points in $P_i \cup \overline{P}_j$. Hence, the problem can be solved by searching for $O(n^2)$ possible bipartitions of P, namely, $(P_i \cup \overline{P}_j, P \setminus (P_i \cup \overline{P}_j))$ for $0 \leqslant i \leqslant j \leqslant n$, and evaluating the widths of the two strips enclosing each part of desired bipartitions.

Let $w_1(i, j) := \text{width}_0(\{p_{i+1}, \ldots, p_j\})$ be the width of the smallest horizontal strip enclosing $j - i$ points $p_{i+1}, \ldots, p_j \in P$, that is, the difference of the y-coordinates of p_{i+1} and p_j. Let $w_2(i, j) := \text{width}_\theta(P_i \cup \overline{P}_j)$ be the width of the smallest strip in orientation θ enclosing $P_i \cup \overline{P}_j = \{p_1, \ldots, p_i, p_{j+1}, \ldots, p_n\}$. Define $w(i, j) := \max\{w_1(i, j), w_2(i, j)\}$. Our task is to minimize $w(i, j)$ over all $0 \leqslant i \leqslant j \leqslant n$. This can be done by evaluating $w_1(i, j)$ and $w_2(i, j)$ for at most $4n$ pairs (i, j) of indices due to the monotonicity of w_1 and w_2. More precisely, observe that

$$w_1(i, j) \leqslant w_1(i, j+1) \quad \text{and} \quad w_1(i, j) \leqslant w_1(i-1, j),$$

while

$$w_2(i, j) \geqslant w_2(i, j+1) \quad \text{and} \quad w_2(i, j) \geqslant w_2(i-1, j)$$

by definition. Hence, our algorithm initially sets $i = j = 0$ and repeatedly increases j by one until it holds that $w_1(0, j) \leqslant w_2(0, j)$ and $w_1(0, j + 1) \geqslant w_2(0, j + 1)$. Then for each $i = 1, \ldots, n$ in this order, it repeatedly increases j by one until it holds that $w_1(i, j) \leqslant w_2(i, j)$ and $w_1(i, j + 1) \geqslant w_2(i, j + 1)$ for the current i. This way, our algorithm probes at most $4n$ pairs (i, j). For a given pair (i, j), in $O(1)$ time we can evaluate $w_1(i, j)$ by definition and $w_2(i, j)$ by Lemma 1. We thus conclude the following.

▶ **Theorem 2.** *Given a set P of n points and two orientations $\theta, \phi \in [0, \pi)$, the two-line center problem where the resulting lines have orientations θ and ϕ can be computed in $O(n \log n)$ time, or in $O(n)$ time, provided P is sorted in orientation either θ or ϕ.*

3 One fixed orientation

In this section, we solve the second constrained problem: given a fixed orientation ϕ, find two strips of minimum width whose union encloses P such that one of the two strips is in orientation ϕ. Throughout this section, a pair of two such strips (σ_1, σ_2), where σ_1 is in orientation ϕ, will be simply called a *constrained two-strip*, and assume that $\phi = 0$.

To find a constrained two-strip of minimum width enclosing P, one could make use of a data structure for the dynamic width maintenance [13, 17]. Observe that there are $O(n^2)$ possible bipartitions of P induced by a constrained two-strip (σ_1, σ_2) since there are $O(n^2)$ distinct subsets of P that can be enclosed by a horizontal strip σ_1. This approach, however, does not seem to avoid a quadratic running time, since the point set we would maintain undergoes $\Theta(n^2)$ updates. Another common approach is to apply known techniques, such as the parametric search [5], the expander-based method [26], or the one based on a sorted matrix [22]. These techniques also require at least quadratic time overhead.

Despite this difficulty, we present a near-linear $O(n \log^3 n)$-time algorithm based on our $O(n \log^2 n)$-time decision algorithm and Chan's structure of dynamic width maintenance [13]. Note that the decision problem can be solved in $O(n \log^3 n)$ time by a direct application of the machinery of Agarwal and Sharir [3]. In the following, we show how to shave another logarithmic factor, while still using the data structure of Agarwal and Sharir.

3.1 Data structures for dynamic width decision and maintenance

Agarwal and Sharir [3] showed that in $O(n \log^3 n)$ time the *offline dynamic width decision problem* can be solved: Given a parameter $\omega > 0$ and a sequence of n insert/delete operations on a set S of points, initially consisting of at most n points, determine whether there is any moment such that $\mathrm{width}(S) \leqslant \omega$ during the n updates on S. Their algorithm builds a segment tree based on the life-spans of the points, that is, the time intervals in which each point is a member of S, and traverse it with a secondary data structure $\mathcal{D}$ that maintains necessary information about the width of the current S using linear space.

The data structure $\mathcal{D}$ consists of two balanced binary search trees that store the edges of the convex hull $\mathrm{conv}(S)$, ordered by their orientations, and maintains a certain collection of invariants, which suffice to decide in $O(1)$ time whether or not $\mathrm{width}(S) \leqslant \omega$ for the current set S. Agarwal and Sharir showed that how to update $\mathcal{D}$ per insertion of a point into S, and also how to undo the latest insertion, recovering the structure $\mathcal{D}$ to the status before the latest insertion. Summarizing, we have:

▶ **Lemma 3** (Agarwal and Sharir [3]). *Suppose the data structure $\mathcal{D}$ with a parameter ω has been built on a set S of n points. Then, we can decide whether or not $\mathrm{width}(S) \leqslant \omega$ in $O(1)$ time, and $\mathcal{D}$ can be maintained in $O(\log^2 n)$ worst-case time for the following updates: inserting a point to S and undoing the latest insertion.*

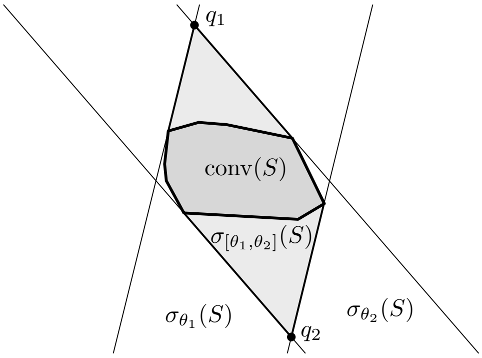


Figure 1 Illustration of $\sigma_{[\theta_1,\theta_2]}(S) = \text{conv}(S \cup \{q_1, q_2\})$ (shaded in light gray) when $\theta_1 < \theta_2$.

Chan [13] presented how to exactly maintain width(S) over fully online updates on S. Its amortized time per update is $O(\sqrt{n}\log^3 n)$, based on the following data structure.

▶ **Lemma 4** (Chan [13]). *There is a data structure $\mathcal{W}$ for a set S of n points that supports deletions of points from S and queries of the following kind: given a query point set Q, report* width$(S \cup Q)$*. The total preprocessing and deletion time is $O(n\log^3 n)$ and the query time is $O(|Q|\log^3(n + |Q|))$. The space required for maintaining the structure $\mathcal{W}$ is $O(n\log n)$.*

Though Chan did not discuss the space requirement for his method, it is not difficult to see it as stated above from construction [13]. Note that if the online updates are deletions only, then this results in an $O(n\log^3 n)$-time algorithm that maintains the exact width.

3.2 Orientation-constrained width

Let S be a set of points, and let $\theta_1 \leqslant \theta_2$ be two orientations. Define the $[\theta_1, \theta_2]$*-constrained width* of S to be

$$\text{width}_{[\theta_1,\theta_2]}(S) := \min_{\theta\in[\theta_1,\theta_2]} \text{width}_\theta(S).$$

Note that $\text{width}(S) = \text{width}_{[0,\pi]}(S)$ and $\text{width}_{[\theta,\theta]}(S) = \text{width}_\theta(S) = \text{width}(\sigma_\theta(S))$. Also, define

$$\sigma_{[\theta_1,\theta_2]}(S) := \bigcap_{\theta\in[\theta_1,\theta_2]} \sigma_\theta(S).$$

Note that $\sigma_{[\theta_1,\theta_2]}(S)$ is the convex hull of S and two more points from the boundary of the intersection of two strips $\sigma_{\theta_1}(S)$ and $\sigma_{\theta_2}(S)$. See Figure 1 for an illustration.

▶ **Lemma 5.** *For any finite set S of points, it holds that* $\text{width}_{[\theta_1,\theta_2]}(S) = \text{width}(\sigma_{[\theta_1,\theta_2]}(S))$.

As will be seen later, we are also interested in *orientation-constrained width decision queries*. More precisely, we are given a query interval $[\theta_1, \theta_2] \subseteq [0, \pi)$ of orientations and want to decide whether there exists $\theta \in [\theta_1, \theta_2]$ such that the width of $\sigma_\theta(S)$ is at most ω or, equivalently, whether $\text{width}_{[\theta_1,\theta_2]}(S) \leqslant \omega$. It turns out that the structure $\mathcal{D}$ of Lemma 3 by Agarwal and Sharir is helpful for this type of queries as well, with the aid of Lemma 5.

▶ **Lemma 6.** *Provided the data structure $\mathcal{D}$ on a point set S of n points with parameter ω is available, an orientation-constrained width decision query on S for width ω can be answered in $O(\log^2 n)$ worst-case time using $O(\log n)$ additional space.*

Now, consider two sets S_1 and S_2 of points in the plane that can be separated by a line, that is, $\operatorname{conv}(S_1) \cap \operatorname{conv}(S_2) = \emptyset$. Then, there are exactly two outer common tangent lines ℓ_1 and ℓ_2. Let $\theta_1 \leqslant \theta_2$ be the orientations of ℓ_1 and ℓ_2. We say that S_1 *dominates* S_2 if $\sigma_{[\theta_1,\theta_2]}(S_2) \subseteq \sigma_{[\theta_1,\theta_2]}(S_1)$. By construction, note that either S_1 or S_2 dominates the other. We also mean by the distance between two convex, compact sets A and B, denoted by $d(A, B)$, the minimum length of translation vectors τ such that A and $B + \tau$ have a common point.

▶ **Lemma 7.** *With the above notations, suppose that S_1 dominates S_2. Then, it holds that:*

(i) $\operatorname{width}_{[\theta_1,\theta_2]}(S_1 \cup S_2) = \operatorname{width}_{[\theta_1,\theta_2]}(S_1)$.

(ii) *If* $\operatorname{width}(S_1 \cup S_2) < d(\operatorname{conv}(S_1), \operatorname{conv}(S_2))$, *then* $\operatorname{width}(S_1 \cup S_2) = \operatorname{width}_{[\theta_1,\theta_2]}(S_1)$.

Proof. Since S_1 dominates S_2, we have

$$S_2 \subseteq \sigma_{[\theta_1,\theta_2]}(S_2) \subseteq \sigma_{[\theta_1,\theta_2]}(S_1).$$

Lemma 5 implies that

$$\sigma_{[\theta_1,\theta_2]}(S_1 \cup S_2) = \sigma_{[\theta_1,\theta_2]}(S_1),$$

so the first statement (i) follows. See Figure 2(a).

Suppose $\operatorname{width}(S_1 \cup S_2) < d(\operatorname{conv}(S_1), \operatorname{conv}(S_2))$. Let $\sigma^* = \sigma_{\theta^*}(S_1 \cup S_2)$ be a minimum-width strip enclosing $S_1 \cup S_2$ whose orientation is θ^*. We claim that $\theta^* \in [\theta_1, \theta_2]$. If this claim is true, then, by definition, we have

$$\sigma_{[\theta_1,\theta_2]}(S_1) = \sigma_{[\theta_1,\theta_2]}(S_1 \cup S_2) \subseteq \sigma_{\theta^*}(S_1 \cup S_2) = \sigma^*,$$

so

$$\operatorname{width}(\sigma_{[\theta_1,\theta_2]}(S_1)) \leqslant \operatorname{width}(S_1 \cup S_2),$$

on one hand. On the other hand, since $S_1 \cup S_2$ is a subset of $\sigma_{[\theta_1,\theta_2]}(S_1 \cup S_2) = \sigma_{[\theta_1,\theta_2]}(S_1)$, we also have

$$\operatorname{width}(\sigma_{[\theta_1,\theta_2]}(S_1)) \geqslant \operatorname{width}(S_1 \cup S_2),$$

and the second statement (ii) is thus proved.

Hence, we are done by proving the claim that $\theta^* \in [\theta_1, \theta_2]$. As σ^* is minimal among those enclosing $S_1 \cup S_2$, its boundary contains three points p, q, r from $S_1 \cup S_2$ such that p and q lie on a common bounding line ℓ of σ^*, r lies on the other bounding line ℓ', and the perpendicular foot r' of r to ℓ lies in between p and q. See Figure 2(b) for an illustration.

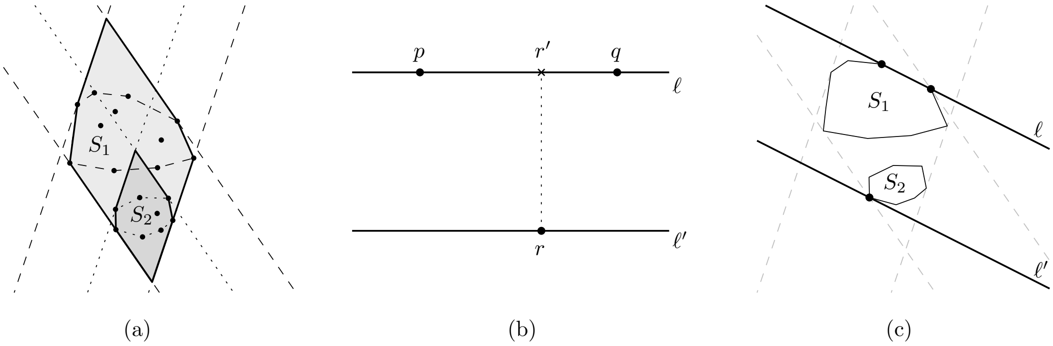


Figure 2 Illustrations to the proof of Lemma 7.

We first exclude the possibility that both p and q belong to a common set, S_1 or S_2, and r to the other. Suppose for a contradiction that, say, $p, q \in S_1$ and $r \in S_2$. By our assumption that $d(\mathrm{conv}(S_1), \mathrm{conv}(S_2)) > \mathrm{width}(S_1 \cup S_2)$, the distance d from r to its perpendicular foot r' is strictly larger than $\mathrm{width}(S_1 \cup S_2)$, while, however, the distance d is also the width of σ^*, a contradiction to the assumption that the width of σ^* determines $\mathrm{width}(S_1 \cup S_2)$.

Now, suppose that $\theta^* \notin [\theta_1, \theta_2]$. Then, observe that both bounding lines ℓ and ℓ' of σ^* cannot intersect a common set, S_1 or S_2; that is, $\ell \cap S_i \neq \emptyset$ if and only if $\ell' \cap S_i = \emptyset$, for $i = 1, 2$. (See Figure 2(c).) This implies that $p, q \in \ell$ belong to one common set, S_1 or S_2, and $r \in \ell'$ belongs to the other set, which is forbidden by the above argument. Thus, we have $\theta \in [\theta_1, \theta_2]$, and the claim is true. ◀

3.3 Decision algorithm

We describe our decision algorithm for a given parameter $\omega > 0$. The points $p_1, \ldots, p_n \in P$ are assumed to be sorted in the y-coordinates, and we let $P_i = \{p_1, \ldots, p_i\}$ and $\overline{P}_i = P \setminus P_i$. Consider a horizontal strip σ_0 of width ω and sweep the plane by translating σ_0 upwards from below. At any moment of this sweeping process, the points $P \setminus \sigma_0$ outside of σ_0 is partitioned into P_i and $\overline{P}_j$ for some $0 \leqslant i \leqslant j \leqslant n$ such that all points of P_i lies below σ_0 and all points of $\overline{P}_j$ lies above σ_0. Note that the indices i and j representing such a separation by σ_0 do not decrease during the process, and it always holds that $d(\mathrm{conv}(P_i), \mathrm{conv}(\overline{P}_j)) > \omega$. Also, by this monotonicity of i and j, observe that $\overline{P}_j$ dominates P_i from the beginning until some moment and P_i dominates $\overline{P}_j$ from that moment to the end. See Figure 3.

We maintain convex hulls, $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$, and the data structure $\mathcal{D}$ with parameter ω on set P_i. Maintaining the convex hulls $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$ can be done in $O(n \log n)$ total time [11, 23]; in our case, updates on P_i and $\overline{P}_j$ are offline. Initially, we have $i = j = 0$, so $\mathrm{conv}(P_i) = \emptyset$, $\mathrm{conv}(\overline{P}_j) = \mathrm{conv}(P)$, and $\mathcal{D}$ is initialized for an empty set P_0.

In the main loop of our decision algorithm, as σ_0 moves upwards, we do nothing until P_i becomes dominating $\overline{P}_j$ while we maintain the data structure $\mathcal{D}$ by inserting relevant points. For each pair (i, j) such that P_i dominates $\overline{P}_j$, we decide whether $\mathrm{width}(P_i \cup \overline{P}_j) \leqslant \omega$ or not. If it is the case, we stop the algorithm and report `YES`; otherwise, we proceed the algorithm. The decision is made as follows: We first compute the outer common tangents of $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$ in $O(\log^2 n)$ time using $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$. (See Figure 3.) Let $\theta_1 \leqslant \theta_2$ be the orientations of the two common tangents. We then perform an orientation-constrained width decision query on P_i with query interval $[\theta_1, \theta_2]$. This can be done in $O(\log^2 n)$ time by Lemma 6 using $\mathcal{D}$. If the answer to this query is positive, then we conclude that $\mathrm{width}(P_i \cup \overline{P}_j) \leqslant \omega$; otherwise, we conclude that $\mathrm{width}(P_i \cup \overline{P}_j) > \omega$. The correctness of this decision is guaranteed by the following lemma, which is a direct application of Lemma 7.

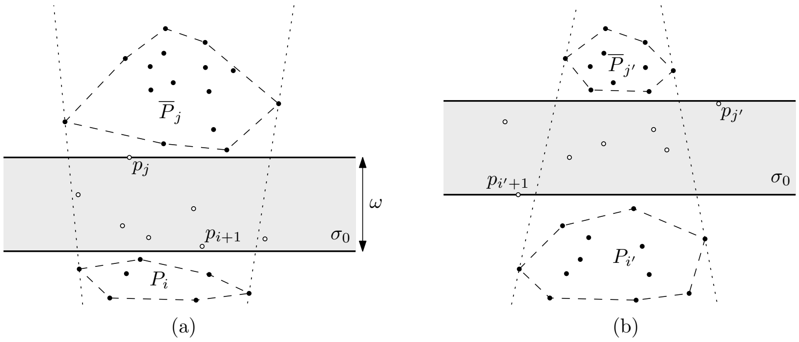


Figure 3 Snapshots of the sweeping process: (a) $\overline{P}_j$ dominates P_i and (b) $P_{i'}$ dominates $\overline{P}_{j'}$.

▶ **Lemma 8.** *For (i, j) such that P_i dominates $\overline{P}_j$ and $d(\mathrm{conv}(P_i), \mathrm{conv}(\overline{P}_j)) > \omega$, we have $\mathrm{width}(P_i \cup \overline{P}_j) \leqslant \omega$ if and only if $\mathrm{width}_{[\theta_1,\theta_2]}(P_i) \leqslant \omega$.*

This way, we check all the pairs (i, j) such that P_i dominates $\overline{P}_j$ during the sweeping process. The other pairs (i, j) such that $\overline{P}_j$ dominates P_i can be handled in a symmetric way by moving the horizontal strip σ_0 downwards. Therefore, we conclude the following.

▶ **Theorem 9.** *Given a set P of n points, $\phi \in [0, \pi)$, and $\omega > 0$, we can decide in $O(n \log^2 n)$ time and $O(n)$ space whether there is a constrained two-strip of width ω.*

3.4 Optimization algorithm

Let w^* be our target optimal width, that is, the minimum width of a constrained two-strip enclosing P. As above, suppose that $p_1, \ldots, p_n \in P$ are sorted in their y-coordinates. Let W_1 be the set of all differences of y-coordinates between two points in P. Since W_1 can be represented by a sorted matrix [19], we can find two consecutive values $w_0, w_1 \in W_1$ such that $w_0 < w^* \leqslant w_1$ in $O(n \log^3 n)$ time using our decision algorithm (Theorem 9) and an efficient selection algorithm for a sorted matrix [20].

At this stage, observe that, for any $w_0 < w < w_1$, the sequence of changes on the sets P_i and $\overline{P}_j$ of points below and above σ_0 is the same as the horizontal strip σ_0 of width w moves upwards. Let X be the set of those pairs (i, j) of indices such that P_i and $\overline{P}_j$ appear as the sets of points below and above σ_0, respectively.

▶ **Lemma 10.** *It holds that*

$$w^* = \min\{w_1, \min_{(i,j)\in X}\{\mathrm{width}(P_i \cup \overline{P}_j)\}\}.$$

Thus, we are done by computing the width of $P_i \cup \overline{P}_j$ for $(i, j) \in X$. Even better, to compute w^* and an optimal two-strip, it suffices to evaluate the exact value of $\mathrm{width}(P_i \cup \overline{P}_j)$ only for those $(i, j) \in X$ such that $\mathrm{width}(P_i \cup \overline{P}_j) < w_1$. Let W be the set of values of $\mathrm{width}(P_i \cup \overline{P}_j)$ that are less than w_1. Below, we show how to compute the set W.

For the purpose, we take any value w with $w_0 < w < w_1$ and simulate the translational sweeping process with the horizontal strip σ_0 of width w in a similar way as done for the decision algorithm. Here, we sweep the plane by moving σ_0 downwards from above. We initialize the data structure $\mathcal{D}$ with parameter $\omega = w_1$ on point set $P_n = P$ by inserting n points $p_1, \ldots, p_n$ in this order, and maintain $\mathcal{D}$ by deleting points in the reversed order to represent P_i, as σ_0 moves downwards. In addition, we initialize the structure $\mathcal{W}$ of Lemma 4 for point set $P_n = P$, and maintain it to store P_i by deleting points in the same order. We also maintain the convex hulls $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$.

During the sweeping process, we only handle those $(i, j) \in X$ such that P_i dominates $\overline{P}_j$, so we stop the process as soon as $\overline{P}_j$ dominates P_i. (Those $(i, j) \in X$ such that $\overline{P}_j$ dominates P_i can be handled in a symmetric way by moving σ_0 upwards.) Consider such a pair (i, j). Our goal is to compute $\mathrm{width}(P_i \cup \overline{P}_j)$ only when it is less than w_1. Note that $d(\mathrm{conv}(P_i), \mathrm{conv}(\overline{P}_j)) \geqslant w_1$. We compute the outer common tangents of $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$ and let $\theta_1 \leqslant \theta_2$ be their orientations. As in the decision algorithm, we test whether or not $\mathrm{width}_{[\theta_1,\theta_2]}(P_i) \leqslant w_1$ by Lemma 6 using $\mathcal{D}$. Lemma 7 implies the following.

▶ **Lemma 11.** *Provided P_i dominates $\overline{P}_j$, $\mathrm{width}(P_i \cup \overline{P}_j) \geqslant w_1$ if $\mathrm{width}_{[\theta_1,\theta_2]}(P_i) > w_1$.*

If it turns out that $\mathrm{width}_{[\theta_1,\theta_2]}(P_i) > w_1$, then we can discard the pair (i, j) and proceed the algorithm by Lemma 11. Otherwise, we compute the exact value of $\mathrm{width}_{[\theta_1,\theta_2]}(P_i)$. If $\theta_1 = \theta_2$, this can be done in $O(\log n)$ time using convex hulls $\mathrm{conv}(P_i)$ and $\mathrm{conv}(\overline{P}_j)$; if $\theta_1 < \theta_2$, by Lemma 5, we have $\text{width}_{[\theta_1,\theta_2]}(P_i) = \text{width}(\sigma_{[\theta_1,\theta_2]}(P_i))$ and $\sigma_{[\theta_1,\theta_2]}(P_i) = \text{conv}(P_i \cup Q)$ where $Q = \{q_1, q_2\}$ consists of two points as described above. (See also Figure 1.) Hence, in this case, we can compute $\text{width}_{[\theta_1,\theta_2]}(P_i)$ in $O(\log^3 n)$ time by Lemma 4 with a query set $Q = \{q_1, q_2\}$ to $\mathcal{W}$. Again, Lemma 7 implies the following.

▶ **Lemma 12.** $\text{width}_{[\theta_1,\theta_2]}(P_i) < w_1$ *if and only if* $\text{width}(P_i \cup \overline{P}_j) < w_1$. *Moreover, if* $\text{width}_{[\theta_1,\theta_2]}(P_i) < w_1$, *then* $\text{width}(P_i \cup \overline{P}_j) = \text{width}_{[\theta_1,\theta_2]}(P_i)$.

By Lemma 12, we have $\text{width}(P_i \cup \overline{P}_j) = \text{width}_{[\theta_1,\theta_2]}(P_i)$ and it is a member of W if and only if the computed value of $\text{width}_{[\theta_1,\theta_2]}(P_i)$ is strictly smaller than w_1.

This way, we can collect the values in W in $O(n \log^3 n)$ time. By Lemma 10, the minimum value in W is w^*, if W is nonempty; or $w^* = w_1$, if $W = \emptyset$. The corresponding two-strip of width w^* can be computed and stored during the execution of the algorithm. Hence, we finally conclude the following.

▶ **Theorem 13.** *Given a set P of n points in the plane and an orientation ϕ, a two-line center for P in which one of the two lines is constrained to be in orientation ϕ can be computed in $O(n \log^3 n)$ time and $O(n \log n)$ space.*

4 Fixed angle of intersection

In this section, we solve the third constrained two-line center problem in which, given a real value $0 \leqslant \beta \leqslant \pi/2$, the difference of the orientations of the two resulting lines is exactly β. Throughout this section, for convenience, a *constrained two-strip* denotes a pair of strips whose orientations differ by β. Let w^* be the minimum width of a constrained two-strip enclosing P. We start by describing optimal configurations.

▶ **Lemma 14.** *There exists a minimum-width constrained two-strip (σ_1, σ_2) enclosing P that falls into one of the following cases:*

(i) *Either $w^* = \text{width}(\sigma_1) > \text{width}(\sigma_2)$ or $w^* = \text{width}(\sigma_2) > \text{width}(\sigma_1)$, and one of the four bounding lines of σ_1 and σ_2 contains two points in P.*

(ii) *It holds that $w^* = \text{width}(\sigma_1) = \text{width}(\sigma_2)$, and each of the four bounding lines of σ_1 and σ_2 contains a point in P.*

In the following, we present an algorithm that runs in $O(n^2\alpha(n) \log n)$ time using $O(n^2)$ space. Our algorithm follows a similar flow as for the second problem, consisting of two phases: (1) find a favorably narrow interval $(w_0, w_1]$ that includes our target width w^*, and (2) proceed the search for w^* with the aid of $(w_0, w_1]$. We first present an efficient decision algorithm, and then describe each of the two phases.

4.1 Decision algorithm

Let $\omega > 0$ be a given parameter, and our goal is to decide whether or not $\omega \geqslant w^*$. Throughout this section, we regard θ as a *direction* from the range $[0, 2\pi)$, taken by modulo 2π, and assume that no three points in P are collinear.

We consider a rotational sweeping process with *fixed width* ω described as follows: Take any point $p \in P$ as a pivot. For direction $\theta \in [0, 2\pi)$, let $\ell(\theta)$ and $\ell^+(\theta)$ be two directed lines in θ such that $\ell(\theta)$ is through p and $\ell^+(\theta)$ is at distance ω to the left of $\ell(\theta)$. We simultaneously rotate both lines $\ell(\theta)$ and $\ell^+(\theta)$ counterclockwise by increasing θ, and consider the strip $\sigma(\theta)$ bounded by the two lines. Taking the second strip $\overline{\sigma}(\theta) := \sigma_{\theta+\beta}(P \setminus \sigma(\theta))$ as

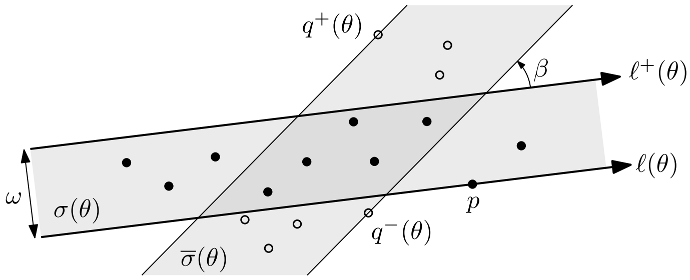


Figure 4 A snapshot at θ of the rotational sweeping process with fixed width ω and pivot p.

the minimum-width strip in orientation $\theta + \beta$ enclosing the rest of points in P, our goal is to decide if there exists $\theta \in [0, 2\pi)$ such that width$(\overline{\sigma}(\theta)) \leqslant \omega$ for some $p \in P$. See Figure 4, in which points in $P \cap \sigma(\theta)$ are depicted by dots and those in $P \setminus \sigma(\theta)$ by small circles.

This sweeping process can be simulated by maintaining the dynamic convex hull conv$(P \setminus \sigma(\theta))$ and its two extreme points $q^-(\theta)$ and $q^+(\theta)$ that define $\overline{\sigma}(\theta)$. Since the number of updates on $P \setminus \sigma(\theta)$ is $O(n)$, it can be done in $O((n + E) \log n)$ time [11], where E denotes the number of changes of the two extreme points $q^-(\theta)$ and $q^+(\theta)$. As will be seen later, $E = O(n\alpha(n))$ and hence the decision can be made in $O(n^2\alpha(n) \log n)$ time.

In order to see why $E = O(n\alpha(n))$ and even to improve the running time, we find it more useful and convenient to discuss the problem in the dual setting. Consider the standard dual transformation that maps each point $r = (a, b) \in \mathbb{R}^2$ into a non-vertical line $r^\star \colon \{y = ax - b\}$, and vice versa. Let $L := \{p^\star \mid p \in P\}$ be the set of n lines dual to each point in P. For a fixed pivot $p = (a, b) \in P$, the trace of $\ell^+(\theta)$ in the dual environment draws a hyperbola $\{y = ax - b \pm \omega\sqrt{1 + x^2}\}$ [5]. We take the upper branch of the hyperbola, denoted by

$$h_p \colon \{y = ax - b + \omega\sqrt{1 + x^2}\},$$

and let $H := \{h_p \mid p \in P\}$. By this choice, we restrict ourselves to considering half the domain of directions, namely $[\pi/2, 3\pi/2)$; the other case can be handled symmetrically by considering the lower branches of the hyperbolas. Note that the dual of $\ell^+(\theta)$ for $\theta \in (\pi/2, 3\pi/2)$ is the intersection point between h_p and vertical line $\{x = \tan(\theta)\}$. Hence, the first strip $\sigma(\theta)$ appears as a vertical segment between p^* and h_p at $x = \tan(\theta)$. Similarly, the dual of the second strip $\overline{\sigma}(\theta)$ is a vertical segment at $x = \tan(\theta + \beta)$ that crosses all but those lines in L intersected by $(\sigma(\theta))^\star$.

For better exposition, we consider an operation that rotates a given line around a given point by angle β. In the dual setting, such a rotation can be performed by the following mapping: for any non-vertical line ℓ and a point $\zeta \in \ell$, we define $\tau_\beta(\zeta; \ell)$ to be the intersection

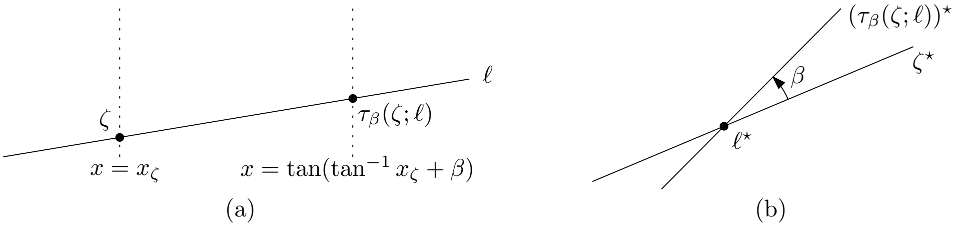


Figure 5 (a) Illustration for the mapping $\tau_\beta(\cdot;\ell)$ on line ℓ and (b) its dual representation.

point between ℓ and the vertical line $\{x = \tan(\tan^{-1}(x_\zeta) + \beta)\}$, where x_ζ is the x-coordinate of ζ. See Figure 5. For a line segment s on ℓ, let $\tau_\beta(s)$ be the segment of ℓ obtained by applying the β-shifting map τ_β to all points on s along ℓ, that is, $\tau_\beta(s) = \bigcup_{\zeta \in s} \tau_\beta(\zeta; \ell)$. Note that $\tau_\beta(s)$ may consist of two half-lines, if the x-coordinates of s are large enough.

Now, consider the pieces of lines in L below $p^\star$ or above h_p. Let S be the set of these segments and half-lines, and $T := \{\tau_\beta(s) \mid s \in S\}$ be the set of shifted segments. We then observe that the two lines bounding $\overline{\sigma}(\theta)$ correspond to the highest and lowest points of the intersection $T \cap \{x = \tan(\theta + \beta)\}$. Thus, $\overline{\sigma}(\theta)$ and its width over $\theta \in (\pi/2, 3\pi/2)$ are determined by the lower and upper envelopes, $\mathcal{L}(T)$ and $\mathcal{U}(T)$, of T. This implies that the number E of changes of the two extreme points $q^-(\theta)$ and $q^+(\theta)$ indeed counts the number of vertices in $\mathcal{L}(T)$ and $\mathcal{U}(T)$, so $E = O(n\alpha(n))$ [30] and the decision problem can be solved in $O(n^2\alpha(n)\log n)$ time. In the following, we improve it to $O(n^2\alpha(n))$ time.

We start with the directed line $\ell(\pi/2)$ through any $p \in P$, and rotate it around the pivot p by increasing θ until it hits another point $p' \in P$. Whenever $\ell(\theta)$ hits another point p', we switch the pivot to p' and continue the rotation around the new pivot p'. Observe that this motion of $\ell(\theta)$ preserves the number k of points in P that lie on $\ell(\theta)$ or on its right side, except some moments when $\ell(\theta)$ contains two points. In the dual setting, the trace of $\ell(\theta)$ is well known as the k-level of the arrangement $\mathcal{A}(L)$ of lines in L. More precisely, for $k = 1, \ldots, n$, the *k-level* of $\mathcal{A}(L)$, denoted by L_k, is the monotone chain consisting of all edges e of $\mathcal{A}(L)$ such that there are exactly $k-1$ lines strictly below any point in the relative interior of e. Similarly, the trace of line $\ell^+(\theta)$ at distance ω from $\ell(\theta)$ is the k-level of the arrangement $\mathcal{A}(H)$ of n hyperbolas in H, denoted by H_k.

Let L_k^- be the region strictly below L_k and H_k^+ be that strictly above H_k. For each $r \in P$ and $1 \leqslant k \leqslant n$, define

$$S_{k,r}^+ := r^\star \cap H_k^+, \quad S_{k,r}^- := r^\star \cap L_k^-, \quad T_{k,r}^+ := \tau_\beta(S_{k,r}^+), \quad \text{and} \quad T_{k,r}^- := \tau_\beta(S_{k,r}^-),$$

and let T_k^+ and T_k^- be the collections of segments and half-lines in $T_{k,r}^+$ and $T_{k,r}^-$, respectively, over all $r \in P$. Our goal is then to compute the envelopes $\mathcal{L}(T_k^+ \cup T_k^-)$ and $\mathcal{U}(T_k^+ \cup T_k^-)$ for all k. As discussed above, these envelopes explicitly describe the changes of the extreme points defining the second strip $\overline{\sigma}(\theta)$ whose orientation is $\theta + \beta$, so we can decide whether $\mathrm{width}(\overline{\sigma}(\theta)) \leqslant \omega$ for some θ in time linear to the total complexity of the envelopes.

Let $\mathcal{L}_k^+ := \mathcal{L}(T_k^+)$, $\mathcal{U}_k^+ := \mathcal{U}(T_k^+)$, $\mathcal{L}_k^- := \mathcal{L}(T_k^-)$, and $\mathcal{U}_k^- := \mathcal{U}(T_k^-)$. We compute these four families of envelopes separately for all k. Then, $\mathcal{L}(T_k^+ \cup T_k^-)$ and $\mathcal{U}(T_k^+ \cup T_k^-)$ can be obtained by merging two envelopes. For our purpose, the following observation is essential.

▶ **Lemma 15.** *For any $1 \leqslant k \leqslant n-1$ and $r \in P$,*

$$T_{k+1,r}^+ \subset T_{k,r}^+ \quad \text{and} \quad T_{k,r}^- \subset T_{k+1,r}^-.$$

Therefore, every point on $\mathcal{L}_k^+$ is below or on $\mathcal{L}_{k+1}^+$; every point on $\mathcal{U}_k^+$ is above or on $\mathcal{U}_{k+1}^+$; every point on $\mathcal{L}_k^-$ is above or on $\mathcal{L}_{k+1}^-$; every point on $\mathcal{U}_k^-$ is below or on $\mathcal{U}_{k+1}^-$.

This naturally suggests us computing each family of envelopes in an incremental way. In the following, we describe how to compute $\mathcal{L}_n^+, \mathcal{L}_{n-1}^+, \ldots, \mathcal{L}_1^+$ in this order. The other three families can also be handled symmetrically and analogously.

Initially, we compute $\mathcal{A}(L)$ and $\mathcal{A}(L \cup H)$, and for each vertex v of $\mathcal{A}(L \cup H)$, we collect its images under τ_β into a set Ξ. More precisely, we use a dictionary structure for Ξ indexed by pairs in $L \times (L \cup H)$, such as an array-based $n \times 2n$ matrix. For each pair (ℓ, ℓ') with $\ell, \ell' \in L$ such that $v = \ell \cap \ell'$ is a vertex of $\mathcal{A}(L \cup H)$, we store $\tau_\beta(v; \ell)$; for (ℓ, h) with $\ell \in L$ and $h \in H$, we store $\tau_\beta(v; \ell)$ for each $v \in \ell \cap h$. We also associate each point $\xi \in \Xi$ with the edge e of $\mathcal{A}(L)$ such that $\xi \in e$, so that given a pair in $L \times (L \cup H)$ we can locate in $O(1)$ time on which edges of $\mathcal{A}(L)$ its relevant points $\xi \in \Xi$ lie. Note that at most two points are stored at each entry of Ξ. The initialization can be done in $O(n^2)$ time using $O(n^2)$ space.

Let $T^+_{n+1} = \mathcal{L}^+_{n+1} = \emptyset$, and suppose $\mathcal{L}^+_{k+1}$ has been correctly computed. Let T^* be the set of segments obtained from $T^+_{k,r} \setminus T^+_{k+1,r}$ for all $r \in P$. We then have $\mathcal{L}^+_k = \mathcal{L}(\mathcal{L}^+_{k+1} \cup T^*)$ by Lemma 15, so can be computed by merging $\mathcal{L}^+_{k+1}$ and $\mathcal{L}(T^*)$. Computing $\mathcal{L}(T^*)$ is done in three steps: specify T^*, compute the arrangement $\mathcal{A}(T^*)$, and extract $\mathcal{L}(T^*)$ from $\mathcal{A}(T^*)$.

To specift T^*, we walk along H_k and H_{k+1} in $\mathcal{A}(L \cup H)$ and find out all intersections $H_k \cap r^\star$ and $H_{k+1} \cap r^\star$ for each $r \in P$. We are then able to extract all segments of $r^\star$ that lie in between H_k and H_{k+1}. For each such segment s, $\tau_\beta(s)$ is a member of T^*. This can be done in $O(m_k + m_{k+1} + n)$ time, where m_i denotes the number of vertices of $\mathcal{A}(L \cup H)$ along H_i. Note that the number of segments in T^* is at most $m_k + m_{k+1}$, since the endpoints of their preimages under the β-shifting map τ_β are all from the vertices along H_k and H_{k+1}.

Note that every $t \in T^*$ is a segment of a line in L, so $\mathcal{A}(T^*)$ is a clipped portion of the entire arrangement $\mathcal{A}(L)$. In addition, the endpoints of t are members of Ξ, so we can find out their exact locations in $\mathcal{A}(L)$ in $O(1)$ time per each. Thus, we can construct $\mathcal{A}(T^*)$ by tracing segments $t \in T^*$ in $\mathcal{A}(L)$ in $O(|T^*| + v_k) = O(m_k + m_{k+1} + v_k)$ time, where v_k denotes the number of vertices of $\mathcal{A}(L)$ we encounter. Note that $\mathcal{A}(T^*)$ consists of exactly $m_k + m_{k+1} + v_k$ vertices and at most $m_k + m_{k+1} + 2v_k$ edges.

As $\mathcal{A}(T^*)$ forms a plane graph, possibly being disconnected, it turns out that its lower envelope $\mathcal{L}(T^*)$ can be obtained in time linear to its complexity. Here, we make use of the following two algorithmic tools: First, a linear-time algorithm for computing the vertical decomposition (or the trapezoidation) of a simple polygon [14,18] can be applied for computing the lower envelope of a connected plane graph.

▶ **Lemma 16.** *Given a connected plane graph G, consisting of m line segments, its lower envelope $\mathcal{L}(G)$ can be computed in $O(m)$ time.*

Second, Asano et al. [9] presented a linear-time algorithm for computing the lower envelope of disjoint line segments, provided their endpoints are sorted. It is not difficult to see that their algorithm also works even if we replace "segments" by "monotone chains."

▶ **Lemma 17** (Asano et al. [9]). *Let $C_1, C_2, \ldots, C_l$ be mutually disjoint monotone chains with m line segments in total. If a sorted list of their endpoints is given, then their lower envelope $\mathcal{L}(\bigcup_i C_i)$ can be computed in $O(m)$ time.*

Back to our problem, we apply Lemma 16 to each connected component of $\mathcal{A}(T^*)$, resulting in disjoint monotone chains $C_1, C_2, \ldots$, whose lower envelope is $\mathcal{L}(T^*)$. Recall that we already have the sorted list of endpoints of T^*, since those endpoints have been obtained by walking along H_k and H_{k+1} in $\mathcal{A}(L \cup H)$ and applying the β-shifting map τ_β, and the map τ_β preserves the order along H_k and H_{k+1}. Hence, we can extract a sorted list of endpoints of the chains C_i in additional $O(m_k + m_{k+1})$ time, which allows us to apply Lemma 17 to obtain $\mathcal{L}(T^*)$. The total time for this third step is proportional to the number of edges in $\mathcal{A}(T^*)$, so $O(m_k + m_{k+1} + v_k)$ time.

Finally, to compute $\mathcal{L}^+_k$, we linearly scan $\mathcal{L}^+_{k+1}$ and $\mathcal{L}(T^*)$, simultaneously. This takes time linear to the total complexity of $\mathcal{L}^+_{k+1}$ and $\mathcal{L}(T^*)$. Note that $\mathcal{L}^+_{k+1}$ consists of $O(|T^+_{k+1}|\alpha(|T^+_{k+1}|)) = O(m_{k+1}\alpha(m_{k+1}))$ edges since it is the lower envelope of line segments [30]. So, the total time we spend to incrementally construct $\mathcal{L}^+_k$ from $\mathcal{L}^+_{k+1}$ for each $1 \leqslant k \leqslant n$ is bounded by $O(n + m_k + m_{k+1}\alpha(m_{k+1}) + v_k)$.

By iterating k from n down to 1, we conclude our decision algorithm.

▶ **Theorem 18.** *Given a set P of n points, an angle β, and a parameter ω, we can decide whether or not $\omega \geqslant w^*$ in $O(n^2\alpha(n))$ time and $O(n^2)$ space.*

4.2 First phase of the optimization algorithm

From now on, we describe our optimization algorithm. Its first phase is done as follows.

Let W_2 be the set of all pairwise distances among points in P. We first obtain two consecutive values $w'_0 < w'_1 \in W_2$ such that $w^* \in (w'_0, w'_1]$. This is easily done in $O(n^2\alpha(n)\log n)$ time by sorting W_2 and performing a binary search on W_2 using our decision algorithm presented in Theorem 18. Next, let W_3 be the set of $n\binom{n}{2}$ values obtained as follows: for any pair $p, q \in P$ with $p \neq q$, collect the distances from each $r \in P$ to the line through p and q. We then find two consecutive values $w''_0 < w''_1 \in W_3$ such that $w^* \in (w''_0, w''_1]$. This can be done in $O(n^2\alpha(n)\log n)$ time by the technique of Glozman et al. [22], again using our decision algorithm. Note that the two-strip of width w''_1 is the best solution of case (i) of Lemma 14. We then choose $w_0 := \max\{w'_0, w''_0\}$ and $w_1 := \min\{w'_1, w''_1\}$, and obtain:

▶ **Lemma 19.** *In $O(n^2\alpha(n)\log n)$ time, we can find two values $w_0 \leqslant w_1$ such that $w_0 < w^* \leqslant w_1$ and no member in $W_2 \cup W_3$ lies in (w_0, w_1).*

4.3 Second phase of the optimization algorithm

For each $p \in P$, let w^*_p be the minimum possible width of constrained two-strips (σ_1, σ_2) such that p lies on the boundary of σ_1. It is obvious that $w^* = \min_{p\in P} w^*_p$. The second phase of our algorithm computes the exact value of w^*_p, if $w^*_p < w_1$; or reports $w^*_p \geqslant w_1$, otherwise. Note that, if $w^*_p < w_1$, then the corresponding optimal two-strip falls in case (ii) described in Lemma 14 by Lemma 19. In the following, let $p \in P$ be fixed and called the *pivot*.

Updates in the sweeping process with fixed width

Before describing the algorithm, we discuss essential ingredients of its correctness, based on Lemma 19. Let $w \in (w_0, w_1)$ be any value. We consider the sweeping process with fixed width w and fixed pivot p, as described at the beginning of Section 4.1. (See Figure 4.) Recall that the first strip $\sigma(\theta)$ is determined by two directed lines $\ell(\theta)$ and $\ell^+(\theta)$ such that $\ell(\theta)$ goes through p and $\ell^+(\theta)$ is at distance w to the left of $\ell(\theta)$, and the second strip $\overline{\sigma}(\theta)$ in orientation $\theta + \beta$ encloses the rest of points in $P \setminus \sigma(\theta)$. Let $P(\theta) := P \cap \sigma(\theta)$. Then, during this sweeping process as θ increases, $P(\theta)$ undergoes a sequence of *updates*. We identify each update by a pair of its involved point $r \in P$ and its *type* determined by one of the four combinations of the following:

- An update is *right* if it happens when $\ell(\theta)$ hits r; or *left* when $\ell^+(\theta)$ hits r
- An update is *leaving* if r is being deleted from $P(\theta)$; or *approaching*, otherwise.

Thus, two updates are the same if their involved points and their types are equal.

Let Υ_w be the set of those updates occurred on $P(\theta)$ during the sweeping process with fixed width w over $\theta \in [0, 2\pi)$. Observe that there are two possibilities for each $r \in P \setminus \{p\}$: By Lemma 19, the distance between r and the pivot p is either at most w_0 or at least w_1. Thus, if r falls in the former case, there are exactly two updates for r in Υ_w whose types are right leaving and right approaching; in the latter case, there are exactly four updates for r in Υ_w with each of the four possible types. This implies that the set Υ_w is invariant under the choice of $w \in (w_0, w_1)$, so we write $\Upsilon = \Upsilon_w$ for any $w \in (w_0, w_1)$.

Fix an arbitrary right leaving update $\upsilon_0 \in \Upsilon$ in which $r_0 \in P \setminus \{p\}$ is involved, and assume that both p and r_0 lie along $\ell(0)$ in this order, that is, p and r_0 lie on the horizontal line $\ell(0)$ and r_0 is to the right of p; this can be easily achieved by a proper rotation of the axes. For $\upsilon \in \Upsilon$ and $w \in (w_0, w_1)$, let $\phi_\upsilon(w) \in [0, 2\pi)$ be the direction at which υ occurs during the sweeping process with fixed width w. From the above discussion, we know that ϕ_υ is a well-defined function from (w_0, w_1) to $[0, 2\pi)$. Lemma 19 implies the following.

▶ **Lemma 20.** *There is no $w \in (w_0, w_1)$ such that $\phi_\upsilon(w) = \phi_{\upsilon'}(w)$ for any two distinct $\upsilon, \upsilon' \in \Upsilon$. Moreover, for each $\upsilon \in \Upsilon$, $\phi_\upsilon(w)$ is either constant if υ is right, continuously increasing if υ is left leaving, or continuously decreasing if υ is left approaching.*

For $w \in (w_0, w_1)$, we consider a total order $\prec_w$ on Υ such that $\upsilon \prec_w \upsilon'$ if and only if $\phi_\upsilon(w) < \phi_{\upsilon'}(w)$. Note that its totality is guaranteed by Lemma 20 and υ_0 is the least element in Υ under $\prec_w$. Lemma 20 further implies that the ordering $\prec_w$ on Υ remains the same over all $w \in (w_0, w_1)$: assuming any swap between $\prec_w$ and $\prec_{w'}$ for $w_0 < w < w' < w_1$, one can face with some $w'' \in (w, w')$ and $\upsilon, \upsilon' \in \Upsilon$ such that $\phi_\upsilon(w'') = \phi_{\upsilon'}(w'')$, due to the continuity of functions ϕ_υ and ϕ_υ, so a contradiction.

Hence, we have a universal total ordering $\prec$ on Υ such that $\prec = \prec_w$ for any $w \in (w_0, w_1)$. Let $\upsilon_0, \upsilon_1, \ldots, \upsilon_{m-1} \in \Upsilon$ be the updates in Υ listed in this order $\prec$, where $m := |\Upsilon|$. For each $0 \leqslant i \leqslant m-1$, let $I_i := \{\phi_{\upsilon_i}(w) \mid w_0 < w < w_1\}$. Lemma 20 implies that I_i consists of a single element if υ_i is a right update; otherwise, I_i forms an open interval if υ_i is a left update. The following summarizes more implications of Lemma 20 about the intervals I_i. Two intervals I and I' are said to be *properly nested* if one includes the other, say $I' \subset I$, in such a way that both endpoints of I' lie in the relative interior of I.

▶ **Lemma 21.** *Two intervals I_i and I_j are never properly nested. If I_i and I_j overlap, then either both of υ_i and υ_j are left leaving or both are left approaching.*

For $0 \leqslant i \leqslant m-1$, let P_i be the resulting set after executing the first $i+1$ updates $\upsilon_0, \ldots, \upsilon_i$ on the subset of points in P lying on or to the left of $\ell(0)$ whose distances to $\ell(0)$ are at most w_0. Note that $P_0 = (\sigma_0 \cap P) \setminus \{r_0\}$ where σ_0 denotes the horizontal strip of width w_0 such that $\ell(0)$ bounds σ_0 from below. Let $Q_i := P \setminus P_i$, and define

$$\omega_i(\theta) := \mathrm{width}_\theta(P_i) \quad \text{and} \quad \overline{\omega}_i(\theta) := \mathrm{width}_{\theta+\beta}(Q_i)$$

for $\theta \in [0, 2\pi)$. Let $r_i \in P \setminus \{p\}$ be the point involved in υ_i.

▶ **Lemma 22.** *For any left leaving update $\upsilon_i \in \Upsilon$, $\omega_{i-1}(\theta) = \mathrm{width}_\theta(\{p, r_i\})$ over $\theta \in I_i$, and is an increasing function over I_i whose infimum and supremum are w_0 and w_1, respectively.*

Description of algorithm

The second phase of our algorithm simulates a similar sweeping process as before, but with the first strip $\sigma(\theta)$ having *variable* width: Let $\omega\colon [0, 2\pi) \to \mathbb{R}$ be a function, which will be specified later. We redefine $\ell^+(\theta)$ to be the line at distance $\omega(\theta)$ to the left of $\ell(\theta)$, and thus $\sigma(\theta)$ to have width $\omega(\theta)$. The second strip $\overline{\sigma}(\theta)$ in orientation $\theta + \beta$ is determined as before to tightly enclose the rest of points in $P \setminus \sigma(\theta)$. Let $\overline{\omega}(\theta) := \mathrm{width}(\overline{\sigma}(\theta))$. This way, the process is completely determined by the width function $\omega(\theta)$.

Our width function $\omega(\theta)$ will be fully determined by when to execute each update $\upsilon_i \in \Upsilon$. For $0 \leqslant i \leqslant m-1$, let ϕ_i be the direction at which the i-th update $\upsilon_i \in \Upsilon$ is executed in our algorithm. We choose the ϕ_i's by the following rules:

- If υ_i is a right update, ϕ_i is the only direction in I_i, that is, $I_i = \{\phi_i\}$.
- If υ_i is a left approaching update, ϕ_i is chosen to be the larger endpoint of I_i.
- If υ_i is a left leaving update, ϕ_i is chosen to be the smallest direction θ such that $\omega_{i-1}(\theta) = \overline{\omega}_{i-1}(\theta)$ over $\theta \in I_i$, if exists; otherwise, ϕ_i is the larger endpoint of I_i.

Note that $\phi_0 = 0$ and let $\phi_m := 2\pi$. It is obvious that either $\phi_i \in I_i$ or ϕ_i is the larger endpoint of I_i. Less obvious is that the resulting ϕ_i's indeed obey the ordering $\prec$ of Υ.

▶ **Lemma 23.** *It holds that* $0 = \phi_0 < \phi_1 \leqslant \phi_2 \leqslant \cdots \leqslant \phi_{m-1} \leqslant \phi_m = 2\pi$*.*

The function $\omega(\theta)$ is then set up as follows: $\omega(0) := w_0$ and $\omega(\theta) := \max\{w_0, \omega_i(\theta)\}$ for $\theta \in (\phi_i, \phi_{i+1}]$ and $0 \leqslant i \leqslant m-1$. We then obtain a conditional correctness of our algorithm.

▶ **Lemma 24.** *Suppose* $w_p^* < w_1$*, and let* θ^* *and* $\theta^* + \beta$ *be the directions of the bounding lines of a corresponding two-strip of width* w_p^* *such that the pivot* p *lies on the right bounding line of direction* θ^**. If* $\theta^* \notin I_i$ *for all left approaching updates* $\upsilon_i \in \Upsilon$*, then there is a left leaving update* $\upsilon_j \in \Upsilon$ *such that* $\theta^* = \phi_j \in I_j$ *and* $w_p^* = \omega(\theta^*) = \omega_{j-1}(\theta^*) = \overline{\omega}_{j-1}(\theta^*) = \overline{\omega}(\theta^*)$*.*

Thus, we can compute w_p^* and its corresponding two-strip by checking each ϕ_i such that $\phi_i \in I_i$ and $\upsilon_i \in \Upsilon$ is a left leaving update, provided the condition of Lemma 24 is satisfied. The other case, where w_p^* is *not* determined by left leaving updates, can be handled by a *reversed* sweeping process that rotates $\sigma(\theta)$ clockwise by decreasing θ from 2π to 0; note that in this reversed process each approaching update becomes a leaving update, and vice versa.

Now, the detailed implementation is presented. Simulating the sweeping process with function $\omega(\theta)$ can be done by maintaining a dynamic set Q, representing $P \setminus \sigma(\theta)$, and its convex hull conv(Q). First, we compute the updates $\upsilon_0, \upsilon_1, \ldots, \upsilon_{m-1} \in \Upsilon$ together with their intervals I_i, and also precompute ϕ_i for all right updates and left approaching updates $\upsilon_i \in \Upsilon$. Initially, $Q = Q_0$ and $Q = Q_i$ while we are in $\theta \in (\phi_i, \phi_{i+1}]$ for each $0 \leqslant i \leqslant m$. We also maintain the two extreme points of Q that determine $\overline{\sigma}(\theta)$: this can be done by two types of queries on conv(Q), finding two tangents of conv(Q) in a given direction and finding the next extreme point of conv(Q) neighboring the current one. Each of these convex hull queries can be answered in $O(\log n)$ amortized time [11].

While we rotate $\sigma(\theta)$ as increasing θ, we execute updates $\upsilon_i \in \Upsilon$ at $\theta = \phi_i$ if ϕ_i has already been computed. Recall that only the execution times ϕ_i of left leaving update υ_i are not precomputed, so they are evaluated during the sweeping process: Suppose the current direction θ lies in I_i for a left leaving update υ_i and the first $j \leqslant i$ updates $\upsilon_0, \ldots, \upsilon_{j-1}$ have already been executed, that is, $Q = Q_{j-1}$ currently at θ and $\overline{\omega}(\theta) = \overline{\omega}_{j-1}(\theta)$. At this moment θ, note that $\theta \in I_j \cap I_i$ and υ_j is also a left leaving update by Lemma 21. Hence, Lemma 22 implies that $\omega(\theta) = \omega_{j-1}(\theta) = \text{width}_\theta(\{p, r_j\})$. We then solve the equation $\omega_{j-1}(\varphi) = \overline{\omega}_{j-1}(\varphi)$. Since the two functions ω_{j-1} and $\overline{\omega}_{j-1}$ are sinusoidal over a range in which the two extreme points of Q_{j-1} do not change [10], this can be done in time proportional to the number of such changes while $Q = Q_{j-1}$. As soon as we find a solution $\phi \in I_j$ such that $\omega_{j-1}(\phi) = \overline{\omega}_{j-1}(\phi)$, we know that $\phi_j = \phi$ by our rules; otherwise, ϕ_j is chosen to be the larger endpoint of I_j.

Since $m = O(n)$, the overall time we spend is bounded by $O(n \log n + E \log n)$, where E denotes the number of changes of the extreme points of Q that define the second strip $\overline{\sigma}(\theta)$. In the dual setting, as done for the decision algorithm, those changes correspond to the vertices of the lower and upper envelopes of $O(n)$ line segments, so we have $E = O(n\alpha(n))$ [30]. By iterating pivots $p \in P$, the second phase of the algorithm can be implemented in $O(n^2\alpha(n) \log n)$ total time.

Therefore, we conclude the following result.

▶ **Theorem 25.** *Given a set* P *of* n *points and a parameter* $\beta \in [0, \pi/2]$*, the two-line center problem with a constraint that the resulting two lines should make an angle of* β *can be solved in* $O(n^2\alpha(n) \log n)$ *time using* $O(n^2)$ *space.*

References

1 Pankaj K. Agarwal, Cecilia M. Procopiuc, and Kasturi R. Varadarajan. A $(1+\varepsilon)$-approximation algorithm for 2-line-center. *Computational Geometry: Theory and Applications*, 26:119–128, 2003. doi:10.1016/S0925-7721(03)00017-8.

2 Pankaj K. Agarwal, Cecilia M. Procopiuc, and Kasturi R. Varadarajan. Approximation algorithms for a k-line center. *Algorithmica*, 42(3):221–230, 2005. doi:10.1007/s00453-005-1166-x.

3 Pankaj K. Agarwal and Micha Sharir. Off-line dynamic maintenance of the width of a planar point set. *Computational Geometry: Theory and Applications*, 1:65–78, 1991. doi:10.1016/0925-7721(91)90001-U.

4 Pankaj K. Agarwal and Micha Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proceedings of the 2nd Annuual ACM-SIAM Symposium on Discrete Algorithms (SODA 1991)*, pages 449–458. SIAM, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127865.

5 Pankaj K. Agarwal and Micha Sharir. Planar geometric location problems. *Algorithmica*, 11(2):185–195, 1994. doi:10.1007/BF01182774.

6 Pankaj K. Agarwal and Micha Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete & Computational Geometry*, 16(4):317–337, 1996. doi:10.1007/BF02712871.

7 Taehoon Ahn and Sang Won Bae. Constrained two-line center problems, 2024. arXiv:2409.13304.

8 Taehoon Ahn, Chaeyoon Chung, Hee-Kap Ahn, Sang Won Bae, Otfried Cheong, and Sang Duk Yoon. Minimum-width double-slabs and widest empty slabs in high dimensions. In J.A. Soto and A. Wiese, editors, *Proceedings of the 16th Latin American Theoretical Informatics (LATIN 2024), Part I*, volume 14578 of *LNCS*, pages 303–317, 2024. doi:10.1007/978-3-031-55598-5_20.

9 Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of dijoint polygons. *Algorithmica*, 1:49–63, 1986. doi:10.1007/BF01840436.

10 Sang Won Bae. Minimum-width double-strip and parallelogram annulus. *Theoretical Computer Science*, 833:133–146, 2020. doi:10.1016/j.tcs.2020.05.045.

11 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Symposium on Foundations of Compututer Science (FOCS 2002)*, pages 617–626, 2002. doi:10.1109/SFCS.2002.1181985.

12 T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12(1-2):67–85, 2002. doi:10.1142/S0218195902000748.

13 Thimothy M. Chan. A fully dynamic algorithm for planar width. *Discrete & Computational Geometry*, 30:17–24, 2003. doi:10.1007/s00454-003-2923-8.

14 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991. doi:10.1007/BF02574703.

15 Chayoon Chung, Taehoon Ahn, Sang Won Bae, and Hee-Kap Ahn. Parallel line centers with guaranteed separation. In *Proceedings of the 35th Canadian Conference on Computational Geometry (CCCG 2023)*, pages 153–160, 2023.

16 Arun Kumar Das, Sandip Das, and Joydeep Mukherjee. Approximation algorithms for orthogonal line centers. *Discrete Applied Mathematics*, 338:69–76, 2023. doi:10.1016/j.dam.2023.05.014.

17 David Eppstein. Incremental and decremental maintenance of planar width. *Journal of Algorithms*, 37:570–577, 2000. doi:10.1006/jagm.2000.1107.

18 Alain Fournier and Delfin Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3:153–174, 1984. doi:10.1145/357337.357341.

19 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *Journal of Computer and System Science*, 24:197–208, 1982. doi:10.1016/0022-0000(82)90048-4.

20 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984. doi:10.1137/0213002.

21 Alex Glozman, Klara Kedem, and Gregory Shpitalnik. On some geometric selection and optimization problems via sorted matrices. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS 1995)*, volume 955 of *LNCS*, pages 26–37, 1995. doi:10.1007/3-540-60220-8_48.

22 Alex Glozman, Klara Kedem, and Gregory Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Computational Geometry: Theory and Applications*, 11(1):17–28, 1998. doi:10.1016/S0925-7721(98)00017-0.

23 John Hershberger and Subhash Suri. Off-line maintenance of planar configurations. *Journal of Algorithms*, 21:453–475, 1991. doi:10.1006/jagm.1996.0054.

24 Jerzy Jaromczyk and Miroslaw Kowaluk. The two-line center problem from a polar view: A new algorithm and data structure. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS 1995)*, volume 955 of *LNCS*, pages 13–25, 1995. doi:10.1007/3-540-60220-8_47.

25 Matthew J. Katz, Klara Kedem, and Michael Segal. Constrained square-center problems. In *Proceedings of the 6th Scandinavian Workshop on Algorithm Theory (SWAT 1998)*, volume 1432, pages 95–106. Springer, 1998. doi:10.1007/BFb0054358.

26 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997. doi:10.1137/S0097539794268649.

27 Sang-Sub Kim, Sang Won Bae, and Hee-Kap Ahn. Covering a point set by two disjoint rectangles. *International Journal of Computational Geometry & Applications*, 21(3):313–330, 2011. doi:10.1142/S0218195911003676.

28 Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1(5):194–197, 1982. doi:10.1016/0167-6377(82)90039-6.

29 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985. doi:10.1007/978-1-4612-1098-6.

30 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

31 Godfried T. Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of the 2nd Mediterranean Electrotechnical Conference (IEEE MELECON 1983)*, 1983.

Dynamic Parameterized Problems on Unit Disk Graphs

Shinwoo An
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Kyungjin Cho
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Leo Jang
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Byeonghyeon Jung
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Yudam Lee
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Eunjin Oh
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Donghun Shin
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Hyeonjun Shin
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Chanho Song
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Abstract

In this paper, we study fundamental parameterized problems such as k-Path/Cycle, Vertex Cover, Triangle Hitting Set, Feedback Vertex Set, and Cycle Packing for *dynamic* unit disk graphs. Given a vertex set V changing dynamically under vertex insertions and deletions, our goal is to maintain data structures so that the aforementioned parameterized problems on the unit disk graph induced by V can be solved efficiently. Although dynamic parameterized problems on general graphs have been studied extensively, no previous work focuses on unit disk graphs. In this paper, we present the first data structures for fundamental parameterized problems on dynamic unit disk graphs. More specifically, our data structure supports $2^{O(\sqrt{k})}$ update time and $O(k)$ query time for k-Path/Cycle. For the other problems, our data structures support $O(\log n)$ update time and $2^{O(\sqrt{k})}$ query time, where k denotes the output size.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Unit disk graphs, dynamic parameterized algorithms, kernelization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.6

Related Version *Full Version*: https://arxiv.org/abs/2409.13403

Funding This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00358505).
Kyungjin Cho: Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00410835).

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 6; pp. 6:1–6:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

For a set V of n points in the plane, the *unit disk graph* of V is the intersection graph of the unit disks of diameter one centered at the points in V, denoted by $\mathsf{UD}(V)$. Unit disk graphs serve as a powerful model for real-world applications such as broadcast networks [28, 29], biological networks [23] and facility location [34]. Due to various applications, unit disk graphs have gained significant attention in computational geometry. Since most of the fundamental NP-hard problems remain NP-hard even in unit disk graphs, the study of NP-hard problems on unit disk graphs focuses on approximation algorithms and parameterized algorithms [4, 6, 17, 21, 37]. From the perspective of parameterized algorithms, the main focus is to design subexponential-time parameterized algorithms for various problems on unit disk graphs. While such algorithms do not exist for general graphs unless ETH fails, lots of problems admit subexponential-time parameterized algorithms for unit disk graphs.

In this paper, we study fundamental graph problems on *dynamic* unit disk graphs. Given a vertex set V that changes under vertex insertions and deletions, our goal is to maintain data structures so that specific problems for $\mathsf{UD}(V)$ can be solved efficiently. This dynamic setting has attracted considerable interest. For instance, the connectivity problem [11, 12], the coloring problem [27], the independent set problem [9, 19], the set cover problem [1], and the vertex cover problem [8] have been studied for dynamic geometric intersection graphs. Here, all problems, except for the connectivity problem, are NP-hard. All previous work on dynamic intersection graphs for those problems study approximation algorithms. However, like the static setting, parameterized algorithms are also a successful approach for addressing NP-hardness in the dynamic setting. There has been significant research on parameterized algorithms for dynamic general graphs [2, 13, 18, 26]. Surprisingly, however, there have been no studies on parameterized algorithms for dynamic unit disk graphs.

In this paper, we initiate the study of fundamental *parameterized* problems on dynamic unit disk graphs. In particular, we study the following five fundamental problems in the dynamic setting. All these problems are NP-hard even for unit disk graphs [14, 25].

- k-Path/Cycle asks to find a path/cycle of G with exactly k vertices,
- k-Vertex Cover asks to find a set S of k vertices s.t. $G \setminus S$ has no edge,
- k-Triangle Hitting Set asks to find a set S of k vertices s.t. $G \setminus S$ has no triangle,
- k-Feedback Vertex Set asks to find a set S of k vertices s.t. $G \setminus S$ has no cycle, and
- k-Cycle Packing asks to find k vertex-disjoint cycles of G.

In the course of vertex updates, we are asked to solve those problems as a query. Except for k-Path/cycle, a query is given with an integer k. On the other hand, our data structure for k-Path/Cycle uses k in the construction time.

Table 1 Summary of our results. The results marked as * support amortized update times, and the others are worst-case update/query times. Except for k-Path/Cycle, no data structure requires k in the construction time; The parameter k is given as a query. Additionally, the data structure for k-Path/Cycle can answer a decision query in constant time.

	Update time	Query time	Space Complexity
k-Path/Cycle*	$2^{O(\sqrt{k})}$	$O(k)$	$O(kn)$
k-Vertex Cover*	$O(1)$	$2^{O(\sqrt{k})}$	$O(n)$
k-Triangle Hitting Set*	$O(1)$	$2^{O(\sqrt{k})}$	$O(n)$
k-Feedback Vertex Set	$O(\log n)$	$2^{O(\sqrt{k})}$	$O(n)$
k-Cycle Packing	$O(\log n)$	$2^{O(\sqrt{k})}$	$O(n)$

Our results. Our results are summarized in Table 1. Note that these are almost ETH-tight. To see this, recall that no problem studied in this paper admits a $2^{o(\sqrt{k})}n^{O(1)}$-time algorithm in the static setting unless ETH fails [16, 20, 22]. Thus for any data structure for these problems on dynamic unit disk graphs with update time $T_u(n,k)$ and query time $T_q(n,k)$, we must have $n \cdot T_u(n,k) + T_q(n,k) = 2^{\Omega(\sqrt{k})}n^{O(1)}$ unless ETH fails. In particular, $n \cdot T_u(n,k) + T_q(n,k)$ is the time for solving the static problem using dynamic data structures; we insert the vertices one by one and then answer the query. In our case, this static running time is $2^{O(\sqrt{k})}n$ for k-Path/Cycle, $2^{O(\sqrt{k})} + O(n)$ for k-Vertex Cover and k-Triangle Hitting Set, and $2^{O(\sqrt{k})} + O(n \log n)$ for k-Feedback Vertex Set and k-Cycle Packing. Interestingly, as by-products, we slightly improve the running times of the best-known static algorithms in [4, 6] for k-Feedback Vertex Set and k-Cycle Packing on unit disk graphs from $2^{O(\sqrt{k})}n^{O(1)}$ to $2^{O(\sqrt{k})} + O(n \log n)$.[1]

A main tool used in this paper is *kernelization*, a technique compressing an instance of a problem into a small-sized equivalent instance called a *kernel*. Kernelization is one of the fundamental techniques used in the field of parameterized algorithms [15]. We use the same framework for all problems, except for k-Path/Cycle: For each update, we maintain kernels for the current unit disk graph. Given a query, it is sufficient to solve the problem on the kernel instead of the entire unit disk graph. Precisely, if the kernel size exceeds a certain bound, we immediately return a correct answer. Otherwise, the kernel size is small, say $O(k)$, and thus we can answer the query by applying the static algorithms on the kernel.

Related work. While dynamic parameterized problems on unit disks have not been studied before, static parameterized algorithms have been widely studied for unit disk graphs. For instance, Fomin et al. [20] presented $2^{O(\sqrt{k}\log k)}n^{O(1)}$-time algorithms for k-Path/Cycle, k-Vertex Cover, k-Feedback Vertex Set, and k-Cycle Packing problems on static unit disk graphs. Subsequently, the running times were improved to $2^{O(\sqrt{k})}(n+m)$ [4, 6, 15, 16, 21], which are all ETH-tight. Additionally, for disk graphs, subexponential time FPT algorithms were studied for k-Vertex Cover and k-Feedback Vertex Set [3, 30].

On the other hand, there are several previous works on dynamic parameterized problems on *general graphs*. Alman et al. [2] presented a dynamic algorithm for k-Vertex Cover supporting $1.2738^{O(k)}$ query time and $O(1)$ amortized update time. They also presented a dynamic algorithm for k-Feedback Vertex Set supporting $O(k)$ query time and $k^{O(k)} \log^{O(1)} n$ amortized update time. Korhonen et al. [26] presented a dynamic algorithm for CMSO testing, parameterized by treewidth. Chen et al. [13] and Dvořák et al. [18] presented a dynamic algorithm for k-Path/Cycle and for MSO testing, respectively, parameterized by treedepth. These algorithms admit *edge* insertions and deletions, and Dvořák et al. [18] also admits *isolated vertex* insertions and deletions, while we deal with vertex insertions and deletions.

An alternative way for dealing with NP-hardness is using approximation. There are numerous works on approximation algorithms for dynamic intersection graphs. For disks, one can maintain $(1+\varepsilon)$-approximation of Vertex Cover [8, 24]. Bhore et al. [9] presented a constant-factor approximation algorithm for the maximum independent set problem for disks. They generalized their result on comparable-sized fat object graphs with the approximation factor depending on a given dimension and fatness parameter. For intervals and unit-squares, Agarwal et al. [1] presented a constant-approximation algorithm for the set cover problem and the hitting set problem.

[1] Moreover, they can be improved further to take $2^{O(\sqrt{k})} + O(n)$ time with a slight modification.

2 Preliminaries

Throughout this paper, we let V be a set of points in the plane, and we let $\mathsf{UD}(V)$ be the *unit disk graph* of V. We interchangeably denote $v \in V$ as a point or as a vertex of $\mathsf{UD}(V)$ if it is clear from the context. For an undirected graph G, we often use $V(G)$ and $E(G)$ to denote the vertex set of G and the edge set of G, respectively. For convenience, we denote the subgraph of G induced by $V(G) \setminus U$ by $G \setminus U$ for a subset U of $V(G)$.

Grid. A grid $\boxplus$ is a partition of the plane into squares (called grid cells) of diameter one. Notice that any two points of V contained in the same grid cell of $\boxplus$ are adjacent in G. Each grid cell $\square$ has its own id: $(\lfloor a/\sqrt{2}\rfloor, \lfloor b/\sqrt{2}\rfloor)$, where a and b are the x- and y-coordinates of a point in $\square$. For any two grid cells $\square, \square'$ with id (x, x') and (y, y'), respectively, we let $d(\square, \square') = \max(|x - x'|, |y - y'|)$. For an integer $\ell > 0$, a grid cell $\square$ is called an *ℓ-neighboring cell* of a grid cell $\square'$ if $d(\square, \square') \leq \ell$. See Figure 1(a) in the full version. We slightly abuse the notation so that $\square$ itself is an ℓ-neighboring cell of $\square$ for all $\ell > 0$. For a point v in the plane, we use $\square_v$ to denote the cell of $\boxplus$ containing v. If it lies on the boundary of a cell, $\square_v$ denotes an arbitrary cell containing v on its boundary.

We do not construct the grid $\boxplus$ explicitly. Instead, we maintain the grid cells containing vertices of V only. We associate each id with a linked list that stores all the vertices of V contained in the grid cell. Once we have the id of $\square$, we fetch the linked list associated with $\square$ in amortized constant time or $O(\log n)$ worst-case time, where n is the number of points of V. More specifically, this can be implemented in $O(1)$ amortized time using dynamic perfect hashing (once true randomness is available) and in $O(\log n)$ worst-case time using 1D range search tree along the lexicographic ordering of the ids. Also, each update of the linked list can be done in the same time bound.

In this paper, we access grid cells only for updating data structures, and then we store the necessary grid cells explicitly in our data structures. Consequently, the update times of our data structures are sometimes analyzed using amortized analysis while the query times are always analyzed using worst-case analysis.

Link-cut tree. When we update data structures, we use link-cut trees. A *link-cut tree* is a dynamic data structure that maintains a collection of vertex-disjoint rooted trees and supports two kinds of operations: a link operation that combines two trees into one by adding an edge, and a cut operation that divides one tree into two by deleting an edge [33]. See Figure 1(b–c) in the full version. Each operation requires $O(\log n)$ time. More precisely, the data structure supports the following query and update operations in $O(\log n)$ time.

- $\mathsf{Link}(u, v)$: If v is the root of a tree and u is a vertex in another tree, link the trees containing v and u by adding the edge between them, making u the parent of v.
- $\mathsf{Cut}(v)$: If v is not a root, this removes the edge between v and its parent, so that the tree containing v is divided two trees containing either v or not.
- $\mathsf{Evert}(v)$: This turns the tree containing v "inside out" by making v the root of the tree.
- $\mathsf{Connected}(u, v)$: This checks if u and v are contained in the same tree.
- $\mathsf{LCA}(u, v)$: This returns the lowest common ancestor of u and v assuming that u and v are contained in the same tree.
- $\mathsf{Root}(u)$: This returns the root of the tree containing u.

All missing proofs and details can be found in the full version. In particular, the data structures and their update/query algorithms for k-Path/Cycle and Vertex Cover can be found in the full version.

3 Dynamic Triangle Hitting Set Problem

In this section, we describe a fully dynamic data structure on the unit disk graph of a vertex set V dynamically changing under vertex insertions and deletions that can answer triangle hitting set queries efficiently. Each query is given with a positive integer k and asks to return a triangle hitting set of $\mathsf{UD}(V)$ of size at most k. This data structure will also be used for Feedback Vertex Set and Cycle Packing in Sections 4 and 5.

Our strategy is to maintain a *kernel* of $(\mathsf{UD}(V), k)$. More specifically, consider the set V_{tri} of vertices contained in triangles of $\mathsf{UD}(V)$. Then $(\mathsf{UD}(V_{\mathsf{tri}}), k)$ is a **yes**-instance if and only if $(\mathsf{UD}(V), k)$ is a **yes**-instance, i.e., it is a kernel of $(\mathsf{UD}(V), k)$. We can show that the size of V_{tri} is $O(k)$ if $(\mathsf{UD}(V), k)$ is a **yes**-instance. Therefore, it is sufficient to maintain V_{tri} for answering queries. However, it seems unclear if V_{tri} can be updated in $O(1)$ time, which is the desired update time. In particular, imagine that a vertex v is inserted to V, and we are to determine if v is contained in a triangle of $\mathsf{UD}(V)$. In the case that two neighboring cells $\square_1$ and $\square_2$ of $\square_v$ contain $\omega(k)$ vertices of V, we need to determine if there are vertices $x_1 \in \square_1$ and $x_2 \in \square_2$ such that x_1, x_2 and v form a triangle of $\mathsf{UD}(V)$.

To overcome this issue, we use a superset of V_{tri} as a kernel. Notice that as long as a subset of V contains V_{tri}, it is a kernel of $(\mathsf{UD}(V), k)$.

Kernel: $\boxplus_{\mathsf{core}}$ and V_{core}. The *core grid cluster*, denoted by $\boxplus_{\mathsf{core}}$, is defined as the union of the 5-neighboring cells of the grid cells containing a vertex of V_{tri} and the 10-neighboring cells of the grid cells containing at least three vertices of V. Then let V_{core} be the set of vertices of V contained in $\boxplus_{\mathsf{core}}$. See Figure 3 in the full version. Note that the degree of every vertex of $V \setminus V_{\mathsf{core}}$ in $\mathsf{UD}(V)$ is $O(1)$. We will use this property for designing the update algorithm.

▶ **Lemma 1.** *The size of V_{core} is $O(k)$ if $\mathsf{UD}(V)$ has at most k vertex-disjoint triangles.*

▶ **Observation 2.** *Given a vertex $v \in V \setminus V_{\mathsf{core}}$, we can compute its neighbors in $\mathsf{UD}(V)$ in $O(1)$ time.*

Query algorithm. Using Lemma 1, we only consider the case that the size of V_{core} is $O(k)$. Otherwise, we return **no**. Then, we can compute the minimum triangle hitting set of $\mathsf{UD}(V)$ in $2^{O(\sqrt{k})}$ time using the standard dynamic programming algorithm observed in [16]. See the full version for details.

Update algorithm. Suppose that we already have V_{core} for the current vertex set V. Imagine that we aim to insert a vertex v to V. Then we need to update V_{core}. For this, it suffices to determine if $\square$ must be contained in $\boxplus_{\mathsf{core}}$ for every 10-neighboring cell $\square$ of $\square_v$. By Observation 3, this takes $O(1)$ time. The deletion of a vertex v from V can be handled analogously in $O(1)$ time.

▶ **Observation 3.** *We can check if a grid cell $\square$ is contained in $\boxplus_{\mathsf{core}}$ in $O(1)$ time.*

▶ **Theorem 4.** *There is an $O(n)$-sized fully dynamic data structure on the unit disk graph induced by a vertex set V supporting $O(1)$ update time that allows us to compute a triangle hitting set of size at most k in $2^{O(\sqrt{k})}$ time.*

4 Dynamic Feedback Vertex Set Problem

In this section, we describe a fully dynamic data structure on the unit disk graph of a vertex set V dynamically changing under vertex insertions and deletions that can answer feedback vertex set queries efficiently. Each query is given with a positive integer k and asks to return

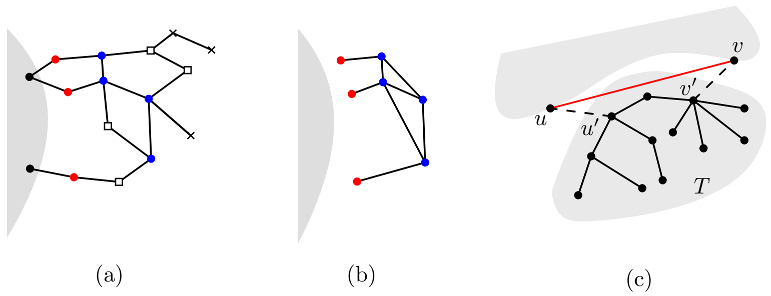


Figure 1 (a) Illustration of $\mathsf{UD}(V)$. The vertices of $\mathsf{UD}(V)$ in V_{core}, the boundary vertices, and the non-boundary vertices in M are marked by the black, red, and blue vertices, respectively. The removed vertices and the contracted vertices in the construction of M are marked by cross vertices and boxes, respectively. (b) Illustration of M. (c) In the construction of M, we are left with the induced path between u and v, which will be contracted to the red edge in M. The two end edges of the path compose the bridge set of T.

a feedback vertex set of $\mathsf{UD}(V)$ of size at most k. As a data structure, we use the core grid cluster $\boxplus_{\mathsf{core}}$ introduced in Section 3. In addition to this, we design a new data structure, which will also be used for Cycle Packing in Section 5.

4.1 Data Structure

Note that a cycle of $\mathsf{UD}(V)$ might contain a vertex lying outside of $\boxplus_{\mathsf{core}}$. Thus we need to consider the part of $\mathsf{UD}(V)$ lying outside of $\boxplus_{\mathsf{core}}$. Since the complexity of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ can be $\Theta(n)$ in the worst case, we cannot afford to look at all such vertices to handle a query. Instead, we maintain a minor M of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ of complexity $O(k)$, which will be called the *skeleton* of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$, such that the graph obtained by gluing $\mathsf{UD}(V_{\mathsf{core}})$ and M has a feedback vertex set of size k if and only if $\mathsf{UD}(V)$ has a feedback vertex set of size k. Then it suffices to look at V_{core} and M to handle a query. Given an update of V, we need to update both V_{core} and M. Since M is a minor of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$, some vertices of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ do not appear in M. To handle the update of V_{core} and M efficiently, we construct an auxiliary data structure $\mathcal{T}$, which maintains the vertices of $\mathsf{UD}(V)$ not appearing in M efficiently.

Skeleton M of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$. Given a query, we will glue M and $\mathsf{UD}(V_{\mathsf{core}})$ together. For this purpose, we need to keep all vertices of $V \setminus V_{\mathsf{core}}$ adjacent to a vertex of V_{core} in the construction of M. We call such a vertex a *boundary vertex.* We let M be the graph obtained from $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ by removing non-boundary vertices with the degree at most one in M repeatedly, and then by contracting every maximal induced path consisting of only non-boundary vertices into a single edge. Note that the resulting graph M is a minor of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$. Each vertex of M corresponds to a vertex of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ of degree at least three or a boundary vertex. Furthermore, each edge of M corresponds to an edge of $\mathsf{UD}(V)$ or an induced path of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$. See Figure 1(a–b). Note that M is planar since every triangle-free disk graph is planar [10].

▶ **Lemma 5.** *If $(\mathsf{UD}(V), k)$ is a* ***yes****-instance of* Feedback Vertex Set, *$|V(M)| = O(k)$.*

Contracted forest $\mathcal{T}$ concerning M. We will see that it suffices to maintain V_{core} and M for the query algorithm. However, to update M efficiently, we need a data structure for the subgraph of $\mathsf{UD}(V)$ induced by $V \setminus (V_{\mathsf{core}} \cup V(M))$. Note that the subgraph is a forest. We maintain the trees of this forest using the link-cut tree data structure. Let $\mathcal{T}$ denote the link-cut tree data structure. If it is clear from the context, we sometimes use $\mathcal{T}$ to denote the forest itself. Along with the link-cut trees, we associate each tree T of $\mathcal{T}$ with a *bridge set*. An edge of $\mathsf{UD}(V)$ incident to both $V(M)$ and $V \setminus (V_{\mathsf{core}} \cup V(M))$ is called a *bridge*. Then the bridge set of T is the set of bridges incident to T. See Figure 1(c).

▶ **Observation 6.** *Each tree of $\mathcal{T}$ is incident to at most two bridges.*

4.2 Query Algorithm

In this subsection, we show how to compute a feedback vertex set of size k of $\mathsf{UD}(V)$ in $2^{O(\sqrt{k})}$ time using V_{core} and M only. Let G be the graph obtained by gluing $\mathsf{UD}(V_{\mathsf{core}})$ and M. More precisely, the vertex set of G is the union of V_{core} and $V(M)$. There is an edge uv in G if and only if uv is either an edge of $\mathsf{UD}(V_{\mathsf{core}})$, an edge of M, or an edge of $\mathsf{UD}(V)$ between $u \in V_{\mathsf{core}}$ and $v \in V(M)$. Notice that v is a boundary vertex of M in the third case.

We use the algorithm proposed by An and Oh [4]. The algorithm of [4] computes a feedback vertex set of size k of a unit disk graph with n vertices in $2^{O(\sqrt{k})}n^{O(1)}$ time. In our case, G is not necessarily a unit disk graph. Thus we need to modify the algorithm of [4] slightly. The details of our algorithm can be found in the full version, and it concludes the following theorem.

▶ **Theorem 7.** *Given M and V_{core}, we can compute a feedback vertex set of $\mathsf{UD}(V)$ of size k in $2^{O(\sqrt{k})}$ time if it exists.*

4.3 Update Algorithm

In this subsection, we illustrate how to update data structures V_{core}, M, and $\mathcal{T}$ with respect to vertex insertions and deletions. We call $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ the *shell* of $\mathsf{UD}(V)$. Here, we slightly abuse the notion of the skeleton so that we can define additional *special vertices* other than the boundary vertices. Imagine that several vertices of V are predetermined as *special* vertices. The other vertices are called *ordinary* vertices. The skeleton M of the shell of $\mathsf{UD}(V)$ with predetermined special vertices is defined as the minor of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ obtained by removing all degree-1 ordinary vertices of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$ repeatedly and then contracting each maximal induced path consisting of ordinary vertices. Notice that if only the boundary vertices of $\mathsf{UD}(V)$ are set to the special vertices, the two definitions of the skeleton coincide.

Given an update of V, we first update V_{core} accordingly using the update algorithm in Section 3. Recall that the number of vertices newly added to V_{core} or removed from V_{core} is $O(1)$. We are to add the vertices removed from V_{core} to the shell of $\mathsf{UD}(V)$, and remove the vertices newly added to V_{core} from the shell of $\mathsf{UD}(V)$. Additionally, $O(1)$ vertices of $V \setminus V_{\mathsf{core}}$ become boundary vertices (when we handle the deletion operation), and $O(1)$ vertices of $V \setminus V_{\mathsf{core}}$ become non-boundary vertices (when we handle the insertion operation.) These are the only changes in the shell of $\mathsf{UD}(V)$ due to the update of V. Note that we just need to add v to the shell of $\mathsf{UD}(V)$ if v is not in V_{core}. Let S be the shell of $\mathsf{UD}(V)$ before the update. Let M be the skeleton of S we currently maintain, and let $\mathcal{T}$ be the contracted forest concerning M we currently maintain.

In the following, we show how to update M and $\mathcal{T}$ for the change of S. The update algorithm consists of two steps: *push-pop step* and *cleaning step*. In the push-pop step, we set several vertices of S as special vertices. Specifically, the new vertices to be added to S are set as special vertices. The neighbors in S of the vertices to be added to S or to be removed from S are set to special vertices. Then we update M and $\mathcal{T}$ to the skeleton of the new set

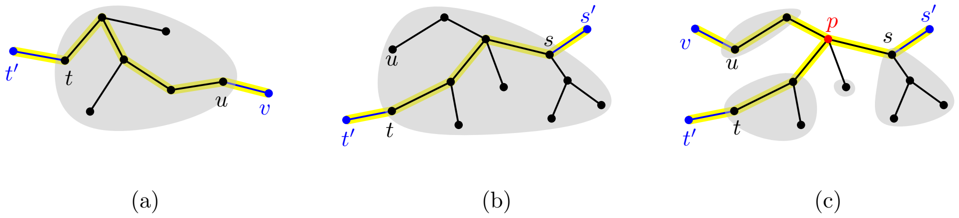


Figure 2 (a) The case that T has exactly one bridge. By the insertion of uv, the path from t' to v, highlighted in yellow, is contracted. (b) The case that T has exactly two bridges before inserting uv. (c) The same case after inserting uv. The vertex p and the edges pv, pt', and ps' are inserted into M.

S and the contracted forest concerning the new skeleton, respectively. In the cleaning step, we make the non-boundary vertices ordinary vertices. Note that this changes the structure of the skeleton as well, and thus we need to update M and $\mathcal{T}$.

4.3.1 Push-Pop Step

Recall that S is the shell of $\mathsf{UD}(V)$ before the update. Notice that the complexity of the new shell of $\mathsf{UD}(V)$ can be $\Theta(n)$ in the worst case. However, the update algorithm in Section 3 determines the vertices and edges added to S and removed from S to obtain the new shell. By construction, the number of such vertices and edges is $O(1)$. Moreover, we can determine the vertices set to the special vertices of S in $O(1)$ time by Observation 2. To update M and $\mathcal{T}$, we add (or remove) the special vertices and their incident edges one by one to (or from) S until S becomes the desired set. Precisely, we add each special vertex v using the *push* subroutine, which adds v into M and updates M and $\mathcal{T}$ accordingly. In the push subroutine, we assume that v was not contained in S. Recall that M is the skeleton of S obtained by removing and contracting some ordinary vertices, not special vertices. And, we remove v using the *pop* subroutine, which removes v from S and update M and $\mathcal{T}$. Note that some special vertices v are already in S. In this case, we cannot make v special using the push subroutine as it assumes that v is not contained in S. For such special vertices v, we first pop v from S and then push it back to S. In this way, we can obtain the skeleton of the new shell and the contracted forest correctly.

Push subroutine. Given the current shell S, we are to add v and its incident edges to S. Let E_v be the set of edges incident to v to be added to S. We can ensure that the other endpoints of the edges of E_v are in S. We first add v and the edges of E_v incident to vertices of M into S. At this moment, it suffices to add v and these edges to M. We do not need to update $\mathcal{T}$. After that, we insert the remaining edges of E_v into S one by one as follows. Note that those edges are incident to vertices of $\mathcal{T}$. Let T be the tree in $\mathcal{T}$ containing another endpoint u of an edge of E_v. Before the insertion of uv, T has its bridge set. There are three cases: T has either exactly zero, one, or two bridges. In particular, in the case that T has exactly one or two bridges we modify M. If T has exactly one bridge, we add an edge into M. And, if T has exactly two bridges, we delete a contracted edge corresponding to an induced path in T and add one vertex and three edges into M. See Figure 2. The details of the procedure for inserting an edge uv of E_v with $u \in \mathcal{T}$ into S are described in the full version.

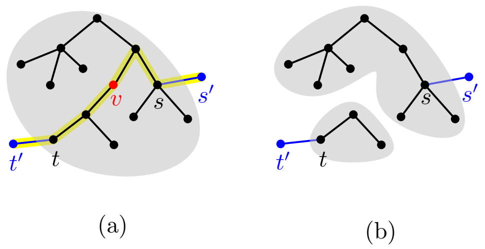


Figure 3 (a) Illustration of pop subroutine in the case that the t-s path contains v. (b) Illustration of the tree T after v is deleted.

Pop subroutine. We are to delete a vertex v and all of its incident edges in S from S. We consider two cases separately: v is in M or $\mathcal{T}$. For the case that v is in M, we simply remove it and its incident edges from M and the bridge sets of $\mathcal{T}$. Specifically, for each edge e incident to v in S, we remove it from M if it is in M. If it is not in M, it is in a bridge set of a tree of $\mathcal{T}$. We remove it from every bridge set. If a bridge set containing e has another edge, then there is a contracted edge of M, and thus we remove it. Then we are done.

Now consider the case that v is in a tree T of $\mathcal{T}$. In this case, the deletion of v from S changes M if and only if T has two bridges tt', ss' with $t, s \in V(T)$ and $t', s' \in V(M)$ such that the t-s path in T contains v. In particular, M has the edge $t's'$, and $t's'$ must be removed from M by the deletion of v from S. See Figure 3. We can check if the t-s path in T contains v by applying $\mathsf{Evert}(v)$ and $\mathsf{LCA}(t, s)$. Then we are to remove v and it incident edges from T and the bridge set of T. Observe that an edge incident to v is in T or the bridge set of T. We rotate T in a way that v becomes the root of T by using $\mathsf{Evert}(v)$, and remove v from T by applying $\mathsf{Cut}(\cdot)$ operations. Then we are given a constant number of child subtrees of v since v is in $V \setminus V_{\mathsf{core}}$. For a child subtree T' of T at v, we insert the bridges of T incident to T' into the bridge set of T'. Note that, given T' and a bridge of T, we can check if T' contains a vertex incident to the bridge using $\mathsf{Connected}(\cdot)$. In this way, we can remove v from $\mathcal{T}$ and the bridge sets, and we can update M accordingly in $O(\log |V|)$ time.

4.3.2 Cleaning Step

So far, we have treated all vertices that can cause some changes due to the update of V and special vertices, and we have computed the skeleton of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$. However, some special vertices should not be considered as special vertices if they are not boundary vertices. To handle this, we need a *cleaning process* to maintain the degree of every vertex in M is at least three except the boundary vertices.

We handle the vertices of M of degree at most two one by one and set each of them as an ordinary vertex if it is a special vertex, as follows. Let v be a vertex we are to handle. First, we can check in $O(1)$ time if v is a boundary vertex using Observation 2. If v is a boundary vertex, we are done. Otherwise, it suffices to handle the case that the degree of v in M is less than three. The update of M is simple: we remove v from M if its degree is one in M, and contract a maximal induced path containing v in M if the degree of v is two in M. Then we need to update $\mathcal{T}$ accordingly as follows. It suffices to merge all trees in $\mathcal{T}$ incident to v together with their bridges incident to v using $\mathsf{Link}(\cdot)$ sequentially. Then the bridge set of the resulting tree is the union of all bridges of merged trees except the ones incident to v. We can handle v in $O(\log |V|)$ time. However, this process might decrease the degree of some other vertex of M of degree in M to less than three. For each such vertex, we remove it or contract a maximal induced path containing it as we did for v.

▶ **Lemma 8.** *The cleaning step takes $O(\log |V|)$ time in total.*

Since both steps can be done in $O(\log |V|)$ time, we have the following lemma.

▶ **Lemma 9.** *Given a vertex update of V, we can update V_{core}, M and $\mathcal{T}$ in $O(\log |V|)$ time.*

▶ **Theorem 10.** *There is an $O(n)$-sized fully dynamic data structure on the unit disk graph induced by a vertex set V supporting $O(\log |V|)$ update time that allows us to compute a feedback vertex set of size at most k in $2^{O(\sqrt{k})}$ time.*

5 Dynamic Cycle Packing Problem

In this section, we describe a fully dynamic data structure on the unit disk graph of a vertex set V dynamically changing under vertex insertions and deletions that can answer cycle packing queries efficiently. Each query is given with a positive integer k and asks to return a set of k vertex-disjoint cycles of $\mathsf{UD}(V)$ if it exists. Here, we use the core grid cluster $\boxplus_{\mathsf{core}}$, the set V_{core} of vertices contained in $\boxplus_{\mathsf{core}}$, the skeleton M of $\mathsf{UD}(V \setminus V_{\mathsf{core}})$, and the contracted forest $\mathcal{T}$ along with the bridge sets. They can be maintained in $O(\log |V|)$ time as shown in Section 4.3. Note that $\mathcal{T}$ and the bridges are the auxiliary data structures for updating M efficiently. In this section, we present a query algorithm for Cycle Packing assuming that we have $\boxplus_{\mathsf{core}}$, V_{core}, and M. The following lemma was given by An and Oh [6].

▶ **Lemma 11.** *If $(\mathsf{UD}(V), k)$ is a **no**-instance for Cycle Packing, then $|V(M)| = O(k)$.*

As in Section 4.2, we first compute the graph G by gluing $\mathsf{UD}(V_{\mathsf{core}})$ and M. More precisely, the vertex set of G is the union of V_{core} and $V(M)$. There is an edge uv in G if and only if uv is either an edge of $\mathsf{UD}(V_{\mathsf{core}})$, an edge of M, or an edge of $\mathsf{UD}(V)$ between $u \in V_{\mathsf{core}}$ and $v \in V(M)$. Notice that v is a boundary vertex of M in the third case.

We use the algorithm proposed by An and Oh [6]. The algorithm of [6] computes a cycle packing of size k of a unit disk graph with n vertices in $2^{O(\sqrt{k})}n^{O(1)}$ time. This algorithm uses the geometric representation of a given graph, and thus the drawing of G is needed. Specifically, we need a drawing of G such that every vertex of G is on its geometric representation and every edge of M does not intersect $\boxplus_{\mathsf{core}}$. For the desired running time, we consider a drawing of G of complexity $\mathrm{poly}(k)$. Since the number of vertices of V_{core} is $O(k)$, it suffices to draw M and the edges between $V(M)$ and V_{core} properly. In Section 5.1, we will see that we can compute a desired drawing of such subgraph of G of complexity $\mathrm{poly}(k)$ in $\mathrm{poly}(k)$ time. Then, we can draw G by merging this drawing with the drawing of $G \setminus M$. Furthermore, in our case, G is not necessarily a unit disk graph. Thus we need to modify the algorithm of [4] slightly. The details of our algorithm can be found in the full version, and it concludes the following theorem.

▶ **Theorem 12.** *Given the core grid cluster $\boxplus_{\mathsf{core}}$ and the skeleton M, Cycle Packing can be solved in $2^{O(\sqrt{k})}$ time.*

5.1 Planar Drawing of the Closure of M

In this subsection, we illustrate how to compute a planar drawing of the closure of M into the plane such that the vertices are drawn on their corresponding points in V, and the drawing does not intersect the boundary of $\boxplus_{\mathsf{core}}$. Here, it suffices to prove the following lemma. In our case, each vertex of the closure of M is prespecified, and each connected region of $\mathbb{R}^2 \setminus \boxplus_{\mathsf{core}}$ is a polygonal domain Σ. By applying the following lemma for each connected region of $\mathbb{R}^2 \setminus \boxplus_{\mathsf{core}}$, we can compute a planar drawing of the closure of M of complexity $\mathrm{poly}(k)$ with the desired properties in $\mathrm{poly}(k)$ time.

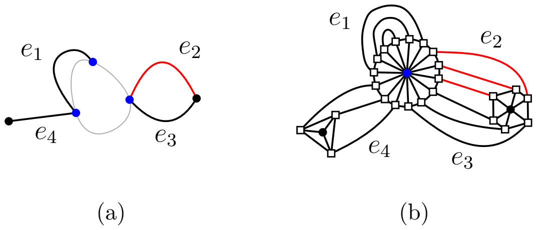


Figure 4 (a) The blue vertices denote the vertices of L on the boundary of a hole. (b) The vertices on a hole are contracted into one blue vertex on the boundary of the hole. And, the black boxes denote the vertices of gadgets. Each edge is copied into three edges. For clarity, the edges corresponding to e_2 are colored in red.

▶ **Lemma 13.** *Any planar graph L admits a planar drawing on a polygonal domain Σ that maps each vertex to its prespecified location and each edge to a polygonal curve with $O(|\Sigma| \cdot |E(L)|^3)$ bends, where $|\Sigma|$ denotes the total complexity of boundaries of Σ. Moreover, we can draw such one in $O(|\Sigma| \cdot |E(L)|^4)$ time if we know the proper ordering of edges incident to each vertex.*

We demonstrate Lemma 13 by Witney's theorem [36, 35] along with the algorithm in [32]. Specifically, Witney's theorem guarantees that a *3-connected* planar graph admits a topologically unique planar embedding.

For clarity, we assume that L is connected, and we demonstrate how to draw a planar drawing of L that maps each vertex to its prespecified location and each edge to a polygonal curve with $O(|\Sigma| \cdot |E(L)|)$ bends in $O(|\Sigma| \cdot |E(L)^2|)$ time. When L is not connected, we can draw the desired planar drawing of L by drawing each component of L using the aforementioned algorithm, sequentially. Precisely, after we compute a drawing of a connected component of L, we add the region containing the drawing into Σ as a hole. We can compute such region in time $O(|\Sigma| \cdot |E(L)|^2)$ time by unifying all faces of the drawing except the outer face. Note that the total complexity of the boundaries of such regions is $O(|\Sigma| \cdot |E(L)|^2)$. Thus we can compute the desired planar drawing of L in $O(|\Sigma| \cdot |E(L)|^4)$ time.

▶ **Lemma 14** (Theorem 1 in [32]). *Every planar graph L admits a planar drawing that maps each vertex to an arbitrarily prespecified distinct location and each edge to a polygonal curve with $O(|V(L)|)$ bends. Moreover, such a drawing can be constructed in $O(|V(L)|^2)$ time.*

We first contract each hole in Σ into a single point and compute the planar drawing of L in a plane without holes using Lemma 14. After that, we recover the holes in L at the prespecified locations. In the recovering step, we want to ensure that the given topological ordering is maintained in the planar drawing of L. To do this, we use Witney's theorem. However, L is not necessarily 3-connected. For this reason, we slightly modify L so that it becomes 3-connected.

Modification for L_{tc} and drawing $\mathcal{D}_{\mathsf{tc}}$. To utilize Witney's theorem and Lemma 14, we obtain a *3-connected* planar graph L_{tc} by modifying L with respect to the holes in the polygonal domain Σ, and we compute a polygonal drawing $\mathcal{D}_{\mathsf{tc}}$ of L_{tc} using Lemma 14. We assume that each vertex in L is the distance at least $0 < \varepsilon < 1$ from any other vertex in L.

We first contract each hole into a single point in the plane. In this way, all vertices of L lying on the boundary of the same hole or the outer boundary of Σ are contracted to a single vertex on the boundary accordingly. Next, we add a *wheel graph* centered on v for each vertex v of L as a gadget. Specifically, the gadget is formed by connecting a single

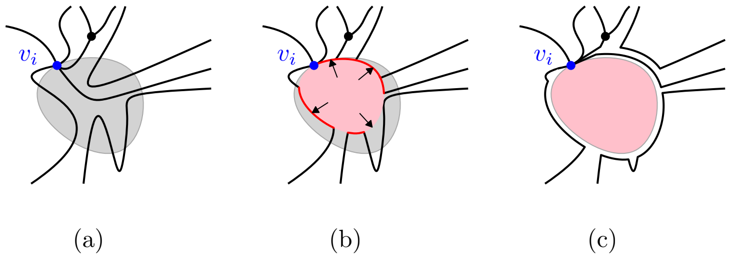


Figure 5 (a) A part of D_{tc} after removing paths. The gray region is a hole A_i, and the blue vertex is v_i. and (b) The pink region is A'_i. While blowing A'_i, we push subcurves on its boundary as the red curves if they intersect A_i. (c) After blowing up A'_i, we perturb the drawing of $\mathcal{D}$ without crossing.

universal vertex to the vertices of a cycle with $3d_v$ vertices of diameter $\varepsilon/100$, where d_v is the degree of v in L. Then we replace each edge uv of L with three edges: we choose three consecutive vertices from the gadget for u and three consecutive vertices from the gadget for v, then we connect them. See Figure 4. We can do this for all edges of L without crossing by maintaining the proper ordering of the edges around each vertex of L. Recall that we have the proper ordering of incident edges at each vertex in L. We denote the result graph as L_{tc}. Note that L_{tc} is a 3-connected planar graph since a wheel graph with at least four vertices is 3-connected.

The number of vertices in L_{tc} is at most $6|E(H)| + |V(H)|$. By Lemma 14, we can compute a planar drawing of L_{tc} in the plane where each edge has at most $O(|E(H)|)$ bends. Furthermore, it is the unique topological embedding by Witney's theorem since L_{tc} is a 3-connected planar graph. We denote the drawing by $\mathcal{D}_{\mathsf{tc}}$. In the following, we recover a polygonal drawing $\mathcal{D}$ of L in Σ.

Recovering $\mathcal{D}$ from $\mathcal{D}_{\mathsf{tc}}$. We recover a drawing $\mathcal{D}$ of L in Σ from $\mathcal{D}_{\mathsf{tc}}$. For each edge uv in L, it corresponds to three paths of length three in L_{tc} connecting the universal vertices of the gadgets for u and v. Among them, except the middle one, we remove two paths from $\mathcal{D}_{\mathsf{tc}}$. While keeping the drawing of $\mathcal{D}_{\mathsf{tc}}$ of the remaining path between the universal vertices for u and v as the drawing of uv of L, we remove the vertices in L_{tc} which are not in L. This process increases the number of bends of each edge by a factor of at least three compared to $\mathcal{D}_{\mathsf{tc}}$. We refer to the obtained drawing as $\mathcal{D}$.

In the following, we recover the boundaries of Σ. Let $A_1, A_2, \ldots,$ and A_ℓ be holes of Σ, Note that the vertices of L on the boundary of A_i are contracted into one vertex in L_{tc}. We refer to the contracted vertex as v_i for each A_i.

For each A_i, we modify $\mathcal{D}$ so that the resulting drawing avoids A_i. Precisely, we blow up a region A'_i, which is initially the point v_i, within a face adjacent to v_i until it becomes A_i. While blowing up A'_i, we push the subcurves of the curves of $\mathcal{D}$ which intersect A'_i onto the boundary of A'_i. Notice that we push such subcurves, avoiding v_i, except the boundary curves of the face containing A'_i. See Figure 5(b). By repeating such a process, we make the interior of A_i empty. Then, by perturbing the drawing $\mathcal{D}$, we can modify the drawing to avoid A_i and crossing. See Figure 5(c). This process increases the bends by a factor of $|\Sigma|$ compared to $\mathcal{D}_{\mathsf{tc}}$.

In the following, we uncontract the vertices v_i's for each A_i's. Let ℓ and ℓ' be short line segments incident to v_i drawn in opposite directions within the face containing A_i so that they intersect $\mathcal{D}$ and A_i only at v_i. See Figure 6(b). Note that the edges in $\mathcal{D}$ incident

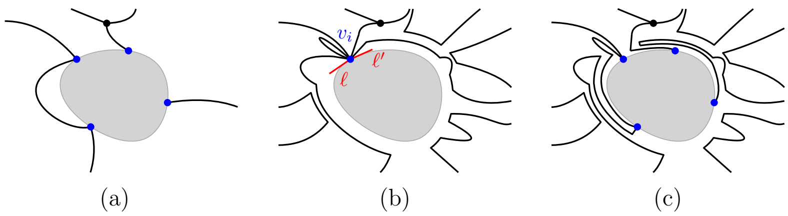


Figure 6 (a) A part of L. The gray region is a hole A_i, and the blue vertices are the vertices of L on the boundary of A_i. (b) A part of drawing D after making A_i empty. The blue vertex is v_i, and the red lines are l and l', respectively. (c) We uncontract v_i into the four blue vertices of L.

to v_i have an endpoint on the boundary of A_i in L, and the ordering is the same as the prespecified ordering along the boundary of A_i in L due to Witney's theorem. We choose the closest incident edge e of v_i to the boundary of A_i, and we uncontract an endpoint u of the edge e' in L corresponding to e contracted into v_i. Precisely, when there is no incident edge of v_i between e and ℓ (or ℓ'), we extend the drawing of e along the boundary of A_i in the counterclockwise direction (or clockwise direction) until u is located at its prespecified location. If both endpoints of e' are contracted into v_i, we uncontract the endpoint preceding in the clockwise order (or counterclockwise order). By repeating such a process, we uncontract all the vertices in L on the boundary A_i at the prespecified location. See Figure 6(c). This process increases the bends of each edge by a factor of at most $|\Sigma|$.

In conclusion, the obtained $\mathcal{D}$ is a polygonal drawing of L in Σ each of which edge has at most $O(|\Sigma| \cdot |E(H)|)$ bends, where $|\Sigma|$ denotes the complexity of the boundary of Σ. Furthermore, the above processes take $O(|\Sigma| \cdot |E(H)|^2)$ time in total. This completes the proof of Lemma 13.

6 Conclusion

In this paper, we initiate the study of fundamental parameterized problems for dynamic unit disk graphs. including k-Path/Cycle, Vertex Cover, Triangle Hitting Set, Feedback Vertex Set, and Cycle Packing. Our data structure supports $2^{O(\sqrt{k})}$ update time and $O(k)$ query time for k-Path/Cycle. For the other problems, our data structures support $O(\log n)$ update time and $2^{O(\sqrt{k})}$ query time, where k denotes the output size. Despite the progress made in this work, there remain numerous open problems. First, can we obtain a trade-off between query times and update times? Second, one might consider other classes of geometric intersection graphs in the dynamic setting such as disk graphs [3, 30], outerstring graphs [7], transmission graphs [5] and hyperbolic unit disk graphs [31]. To the best of our knowledge, there have been no known results on parameterized algorithms for those graph classes.

References

1 Pankaj Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *ACM Transactions on Algorithms (TALG)*, 18(4):1–37, 2022. doi:10.1145/3551639.

2 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Transactions on Algorithms (TALG)*, 16(4):1–46, 2020. doi:10.1145/3395037.

3 Shinwoo An, Kyungjin Cho, and Eunjin Oh. Faster algorithms for cycle hitting problems on disk graphs. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS 2023)*, pages 29–42, 2023. doi:10.1007/978-3-031-38906-1_3.

4 Shinwoo An and Eunjin Oh. Feedback vertex set on geometric intersection graphs. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, pages 47:1–47:12, 2021. doi:10.4230/LIPICS.ISAAC.2021.47.

5 Shinwoo An and Eunjin Oh. Reachability problems for transmission graphs. *Algorithmica*, 84(10):2820–2841, 2022. doi:10.1007/S00453-022-00985-1.

6 Shinwoo An and Eunjin Oh. ETH-tight algorithm for cycle packing on unit disk graphs. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, pages 7:1–7:15, 2024. doi:10.4230/LIPICS.SOCG.2024.7.

7 Shinwoo An, Eunjin Oh, and Jie Xue. Sparse outerstring graphs have logarithmic treewidth. In *32nd Annual European Symposium on Algorithms (ESA 2024)*, pages 10:1–10:18, 2024. doi:10.4230/LIPICS.ESA.2024.10.

8 Sujoy Bhore and Timothy M. Chan. Fully dynamic geometric vertex cover and matching. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.07441.

9 Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, pages 19:1–19:16, 2024. doi:10.4230/LIPICS.SOCG.2024.19.

10 Heinz Breu. *Algorithmic aspects of constrained unit disk graphs.* PhD thesis, University of British Columbia, 1996.

11 Timothy M. Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, pages 36:1–36:13, 2024. doi:10.4230/LIPICS.SOCG.2024.36.

12 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM Journal on Computing*, 40(2):333–349, 2011. doi:10.1137/090751670.

13 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, et al. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 32th ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 796–809. SIAM, 2021.

14 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

15 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for exponential-time-hypothesis–tight algorithms and lower bounds in geometric intersection graphs. *SIAM Journal on Computing*, 49(6):1291–1331, 2020. doi:10.1137/20M1320870.

17 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.

18 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22th Annual European Symposium (ESA 2014)*, pages 334–345, 2014.

19 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

20 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete & Computational Geometry*, 62:879–911, 2019. doi:10.1007/S00454-018-00054-X.

21 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. ETH-tight algorithms for long path and cycle on unit disk graphs. *Journal of Computational Geometry*, 12(2):126–148, 2021. doi:10.20382/JOCG.V12I2A6.

22 Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1563–1575, 2012. doi:10.1137/1.9781611973099.124.

23 Timothy F. Havel, Gordon M. Crippen, Irwin D. Kuntz, and Jeffrey M. Blaney. The combinatorial distance geometry method for the calculation of molecular conformation ii. sample problems and computational statistics. *Journal of Theoretical Biology*, 104(3):383–400, 1983.

24 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

25 Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. doi:10.1137/0211056.

26 Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. Dynamic treewidth. In *Proceedings of the 64th Annual Symposium on Foundations of Computer Science (FOCS 2023)*, pages 1734–1744, 2023.

27 Tomasz Krawczyk and Bartosz Walczak. On-line approach to off-line coloring problems on graphs with geometric representations. *Combinatorica*, 37(6):1139–1179, 2017. doi:10.1007/S00493-016-3414-X.

28 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Unit disk graph approximation. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 17–23, 2004. doi:10.1145/1022630.1022634.

29 Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 69–78, 2003.

30 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract)*. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 2005–2031, 2022. doi:10.1137/1.9781611977073.80.

31 Eunjin Oh and Seunghyeok Oh. Algorithms for Computing Maximum Cliques in Hyperbolic Random Graphs. In *31st Annual European Symposium on Algorithms (ESA 2023)*, pages 85:1–85:15, 2023. doi:10.4230/LIPICS.ESA.2023.85.

32 János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17:717–728, 2001. doi:10.1007/PL00007258.

33 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981)*, pages 114–122, 1981.

34 Da-Wei Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information processing letters*, 28(6):281–286, 1988. doi:10.1016/0020-0190(88)90174-3.

35 Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

36 Hassler Whitney. 2-isomorphic graphs. *American Journal of Mathematics*, 55(1):245–254, 1933.

37 Weili Wu, Hongwei Du, Xiaohua Jia, Yingshu Li, and Scott C.-H. Huang. Minimum connected dominating sets and maximal independent sets in unit disk graphs. *Theoretical Computer Science*, 352(1-3):1–7, 2006. doi:10.1016/J.TCS.2005.08.037.

On the Connected Minimum Sum of Radii Problem

Hyung-Chan An
Yonsei University, Seoul, Republic of Korea

Mong-Jen Kao
National Yang-Ming Chiao-Tung University, Hsinchu, Taiwan

Abstract

In this paper, we consider the study for the *connected minimum sum of radii* problem. In this problem, we are given as input a metric defined on a set of *facilities* and *clients*, along with some *cost parameters*. The objective is to open a subset of facilities, assign every client to an open facilitiy, and connect open facilities using a Steiner tree so that the weighted (by cost parameters) sum of the maximum assignment distance of each facility and the Steiner tree cost is minimized. This problem introduces the *min-sum radii objective*, an objective function that is widely considered in the clustering literature, to the connected facility location problem, a well-studied network design/clustering problem. This problem is useful in communication network design on a shared medium, or energy optimization of mobile wireless chargers.

We present both a constant-factor approximation algorithm and hardness results for this problem. Our algorithm is based on rounding an LP relaxation that jointly models the min-sum of radii problem and the rooted Steiner tree problem. To round the solution we use a careful clustering procedure that guarantees that every open facility has a *proxy client* nearby. This allows a reinterpretation for part of the LP solution as a fractional rooted Steiner tree. Combined with a cost filtering technique, this yields a 5.542-approximation algorithm.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases connected minimum sum of radii, minimum sum of radii, connected facility location, approximation algorithms, Steiner trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.7

Funding This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. RS-2021-II212068, Artificial Intelligence Innovation Hub). This work was supported by the National Science and Technology Council (NSTC), Taiwan, grants 112-2628-E-A49-017-MY3, 113-2628-E-A49-017-MY3, and 113-2634-F-A49-001-MBK.

Acknowledgements The authors made equal contributions. We thank the anonymous reviewers for their helpful comments.

1 Introduction

Connected facility location is a joint optimization problem that combines network design with clustering, and it has wide applications in the design of communication networks [1, 8, 23]. In this problem, we are given as input a metric on a set of nodes (some of which are called *facilities* and some *clients*) in addition to the *opening cost* of each facility and a *connectivity cost parameter* M. The goal is to open some facilities, assign every client to an open facility, and finally connect the open facilities with a Steiner tree whose terminals are the open facilities. The cost of a solution is defined as the total assignment distance between each client and the facility it is assigned to, plus the total opening costs of the open facilities and the cost of the Steiner tree scaled by M. This problem is particularly useful in the design of a communication network where a *central core* is formed by connecting *core nodes* together and individual *endnodes* are assigned to one of the core nodes [1, 8, 23]. There exists an extensive volume of research on this problem: in addition to the problem described

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 7; pp. 7:1–7:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

above [8,10,14,23], a variant where opening a facility itself does not incur opening cost has also been investigated [15,23]. In addition to the facility location problem, connectivity constraints have been introduced in other classical problems including dominating set (see, e.g., [7,13,21,22]).

The facility location problem is a *min-sum* optimization problem, in that it minimizes the *sum* of the assignment distances. Yet, this is not always the objective one is most interested in in practice. For example, when the endnodes are connected via broadcasting on a shared medium, one may be more interested in the longest assignment distance [11]. On the extreme in this direction is the *k-center* problem [17], which minimizes the maximum assignment distance in the *entire* solution. This unfortunately tends to yield a less desirable clustering since the maximum assignment distance by itself determines the objective. In order to avoid the *dissection effect* [16], we can use the sum of radii in lieu of maximum radii as the objective function [6]. Under this objective, however, there exists a trivial optimal solution when there is no distinction between facilities and clients – one can open and assign every node to itself, resulting in the zero sum of radii – and therefore the min-sum radii problem was previously studied usually under a cardinality constraint on the number of open facilities [2,3,6,9,11].

However, the connected facility location problem was little studied under this min-sum radii objective. This paper proposes to study the *connected minimum sum of radii* problem, aiming at addressing this gap. Our problem takes as input the *assignment cost parameter* of each facility in addition to the *connectivity cost parameter M* and a metric on facilities and clients. The goal still is to open some facilities, assign every client to an open facility, and connect the open facilities. The problem however differs from connected facility location in its objective function, which is now defined as the sum of radii, i.e., the sum of the longest assignment distance of each facility, plus the Steiner tree cost connecting the open facilities. The radii and the Steiner tree cost are respectively scaled by the assignment cost parameters and the connectivity cost parameter.

A sample application that well illustrates this problem is wireless charging of sensors. Consider a set of sensors distributed over a region, which are charged by a wireless charger that moves between charging spots to charge near sensors [19,24]. The wireless charging energy is proportional to the maximum distance to a sensor being charged, and the proposed problem well reflects this setting. The connected minimum sum of radii problem also arises when we want to broadcast messages to a set of sensors. Suppose we install a set of mutually connected stations each of which broadcasts messages over the air to nearby sensors. The total communication cost will then depend on the over-the-air broadcast range of each station and their mutual connection cost.

Our results and techniques

In this paper, we propose to study the connected minimum sum of radii problem, present an approximation algorithm for it, and show its NP-hardness. Our main result is the following theorem. While this paper primarily considers the version of the problem that opening a facility itself does not incur a fixed opening cost, Theorem 1 immediately extends to the version with opening cost as well, without affecting the final approximation ratio.

▶ **Theorem 1.** *There is a polynomial-time algorithm that computes a* 5.542*-approximation solution for the connected minimum sum of radii problem.*

The algorithm we present is an LP-rounding algorithm that is partially based on a greedy clustering of fractionally open facilities. Greedy clustering approach was previously used to handle the (non-connected) minimum sum of radii problem [9]. In this paper, we propose that we use a carefully designed new clustering procedure to ensure that each open facility always has a “proxy client” nearby.

After clustering, the LP solution can be reinterpreted as a fractional solution to a rooted Steiner tree instance whose terminals are the proxy clients, at the expense of a slight increase in the cost. This fractional solution is then rounded using any LP-based algorithm with a good approximation ratio for the Steiner tree problem, such as the LP-rounding algorithm of Jain [18] or the primal-dual algorithm of Goemans and Williamson [12]. Finally, to obtain the desired approximation ratio, we compare this solution against a trivial solution that opens a single guessed facility, and output the better between the two solutions.

We will complement the above result by showing that the problem is NP-hard.

▶ **Theorem 2.** *The connected minimum sum of radii problem is NP-hard.*

Organization of this paper

The rest of this paper is organized as follows. In Section 2 we provide a formal definition of the connected minimum sum of radii problem and the notation we will be using throughout this paper. In Section 3 we present our approximation algorithm. We establish the approximation guarantee in Section 4 and present the hardness results in Section 5.

2 Preliminaries

We begin with a formal definition on the connected minimum sum of radii problem. In this problem, we are given a set F of facilities, a set D of clients, a distance metric d defined over $F \cup D$ and two additional parameters $m\colon F \to \mathbb{Q}_{\geq 0}$ and $M \in \mathbb{Q}_{\geq 0}$.

A feasible solution consists of a tuple (S, ρ, T), where $S \subseteq F$ is a subset of facilities, $\rho\colon S \to \mathbb{Q}_{\geq 0}$ is the set of respective radii for the facilities in S such that all clients in D are covered, i.e., for any $j \in D$, there always exists some $i \in S$ such that $d(j,i) \leq \rho_i$, and T is a Steiner tree with terminal set S and Steiner nodes $F \cup D$.

The objective is to minimize the sum of radii of the clusters in S, each weighted by the parameters $m_i\,|_{i\in S}$, plus the total length of the Steiner tree weighted by M, i.e.,

$$\sum_{i\in S} m_i \cdot \rho_i \;+\; \sum_{e\in T} M \cdot d_e.$$

Note that, provided that $M \neq 0$, we may assume without loss of generality that $M = 1$, for otherwise we can scale m_i for all $i \in F$ uniformly. For the rest of this paper we will take this assumption that $M = 1$.

We also note that, our algorithm and the analysis can be modified in a straightforward way to work for the extreme case that $M = 0$.

Notations

We additionally use the following notation in this paper. We use $V := F \cup D$ to denote the set of vertices in the given metric space and $E := \{\, (u,v) \mid u, v \in V \,\}$ to denote the set of possible edges when considering the corresponding metric graph. For any $U \subseteq V$, we use $\delta(U)$ to denote the set of edges in the cut $(U, \bar{U})$ with respect to the metric graph.

For any $i \in F$ and any $r \in \mathbb{Q}_{\geq 0}$, we use $B(i,r)$ to denote the set of clients that belong in the ball centered at i with radius r, i.e,

$$B(i,r) := \{\, j \in D \;\mid\; d(i,j) \leq r \,\}.$$

For each $i \in F$, we use $R_i := \{d(i,j) \mid j \in D\}$ to denote the set of "meaningful" radii for i.

3 Approximation Algorithm

In this section, we present our algorithm for the connected minimum sum of radii problem. Let $\mu \geq 1$ be a parameter to be determined.

Our algorithm starts by guessing a facility $t \in F$ that is opened in an optimal solution $(S^{\mathsf{opt}}, \rho^{\mathsf{opt}}, T^{\mathsf{opt}})$ with the minimum m_t value, i.e., $t = \operatorname{argmin}_{i \in S^{\mathsf{opt}}} m_i$. For each candidate guess t, the algorithm generates two solutions $(S^{\mathrm{I}}_t, \rho^{\mathrm{I}}_t, T^{\mathrm{I}}_t)$ and $(S^{\mathrm{II}}_t, \rho^{\mathrm{II}}_t, T^{\mathrm{II}}_t)$. When this process ends, the one with the smallest cost is output as the approximation solution. In the following we describe how the solutions are generated for each guess $t \in F$. To simplify the notations, the dependency on t will be omitted when there is no ambiguity in the context.

The first solution $(S^{\mathrm{I}}, \rho^{\mathrm{I}}, T^{\mathrm{I}})$ is a trivial one with $S^{\mathrm{I}} := \{t\}$, i.e., t is the only open facility. Naturally, $T^{\mathrm{I}} = \emptyset$ and $\rho^{\mathrm{I}}_t = \max_{j \in D} d(t,j)$ in this solution. The cost of this solution is hence $m_t \cdot \max_{j \in D} d(t,j)$.

To obtain the second solution $(S^{\mathrm{II}}, \rho^{\mathrm{II}}, T^{\mathrm{II}})$, let $F_\mu := \{i \in F \mid m_i \geq \mu\}$. We use the following LP relaxation for a further restricted scenario for which $S^{\mathsf{opt}} \subseteq F_\mu$, i.e., the (unknown) referenced optimal solution only uses facilities in F_μ. This unusual setting will become clear in the analysis.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in F_\mu, r \in R_i} m_i \cdot r \cdot x_{i,r} + \sum_{e \in E} d_e \cdot y_e \\
\text{subject to} \quad & \sum_{i \in F_\mu} z_{i,j} \geq 1, && \forall j \in D, \\
& \sum_{r \in R_i : j \in B(i,r)} x_{i,r} \geq z_{i,j}, && \forall i \in F_\mu, j \in D, \\
& \sum_{e \in \delta(U)} y_e \geq \sum_{i \in F_\mu \cap U} z_{i,j}, && \forall j \in D, U \subseteq V \setminus \{t\}, \qquad (1) \\
& x, y, z \geq 0.
\end{aligned}
$$

We have three sets of indicator variables in the above LP.

- $x_{i,r}$ for each (i,r) pair with $i \in F_\mu$ and $r \in R_i$.
- y_e for each edge $e \in E$.
- $z_{i,j}$ for the assignment of client $j \in D$ to the facility $i \in F_\mu$.

The first constraint requires that any client in D has to be assigned to at least one facility in F_μ. The second constraint demands that, in order for a client j to be assigned to facility i, j must be contained in an opened ball centered at i. The third constraint models the connectivity requirement between the opened facilities via the assignment variables $z_{i,j}$ and the predetermined sink t. Note that the constraints of this LP does not require that t is opened but rather use it to ensure the connectivity between the opened facilities.

Note that the last set of inequalities can be separated by finding a minimum j-t cut. We solve the LP in polynomial time to obtain an optimal fractional solution $(x^\star, y^\star, z^\star)$. In the following we describe our rounding procedure to obtain the second solution $(S^{\mathrm{II}}, \rho^{\mathrm{II}}, T^{\mathrm{II}})$. The rounding procedure consists of two parts. In the first part, we select a set of facilities along with their respective radii to be opened. In the second part, we compute a Steiner tree for the opened facilities.

Opening facilities

Let $\mathcal{B}_0 := \{(i,r) \mid x^\star_{i,r} > 0\}$ be the support of $x^\star$. Let $\mathcal{G}$ be a bipartite graph with partite sets $\mathcal{B}_0$ and D, where $(i,r) \in \mathcal{B}_0$ and $j \in D$ are adjacent if and only if $j \in B(i,r)$. For any $j \in D$ and any $B^\star \subseteq \mathcal{B}_0$, let $\Delta_{\mathcal{B}^\star}(j)$ denote the minimum distance between j and any vertex in $B^\star$: i.e.,

$$\Delta_{\mathcal{B}^\star}(j) := \min\{|P| \mid P \text{ is a path in } \mathcal{G} \text{ between } j \text{ and some } y \in \mathcal{B}^\star\},$$

where $|P|$ denotes the number of edges on P. Note that we define $\Delta_\emptyset(j) := +\infty$ for all $j \in D$.

Algorithm 1 Determining open facilities and their proxy clients.

```
1: $S^{\mathrm{II}} \leftarrow \emptyset$; $\mathcal{B}^\star \leftarrow \emptyset$
2: while $\exists j \in D$ with $\Delta_{\mathcal{B}^\star}(j) \geq 5$ do
3:     $\bar{\mathcal{B}} := \{(i,r) \in \mathcal{B}_0 \mid$ there exists some $j$ that is adjacent to $(i,r)$ and $\Delta_{\mathcal{B}^\star}(j) \geq 5\}$
4:     $(i^\star, r^\star) \in \arg\max_{(i,r)\in\bar{\mathcal{B}}} r$
5:     let $\pi_{i^\star}$ be some $j \in D$ such that $(i,r)$ and $j$ are adjacent and $\Delta_{\mathcal{B}^\star}(j) \geq 5$
6:     $\mathcal{B}^\star \leftarrow \mathcal{B}^\star \cup \{(i^\star, r^\star)\}$; $S^{\mathrm{II}} \leftarrow S^{\mathrm{II}} \cup \{i^\star\}$; $\rho^{\mathrm{II}}_{i^\star} = 3r^\star$
```

Consider Algorithm 1 that returns S^{II}, ρ^{II}, and $\{\pi_i\}_{i\in S^{\mathrm{II}}}$. It additionally maintains $\mathcal{B}^\star$, which denotes the set of (i,r) pairs to be rounded up, and π_i for each $i \in S^{\mathrm{II}}$ which denotes the representative *proxy client* we pick for facility i. It is to ensure the existence of these proxy clients why we use Algorithm 1 as opposed to a simple greedy clustering.

Initially, $S^{\mathrm{II}} := \emptyset$ and $\mathcal{B}^\star := \emptyset$. In each iteration, the algorithm considers the set of (i,r) pairs in $\mathcal{B}_0$ that are adjacent to some client $j \in D$ in $\mathcal{G}$ with $\Delta_{\mathcal{B}^\star}(j) \geq 5$. Among all such (i,r) pairs, the algorithm picks the one with the largest r. Let the pair be (i^*, r^*) and let π_{i^*} be the witness client with $j \in D$ with $\Delta_{\mathcal{B}^\star}(j) \geq 5$.

The algorithm puts i^* in S^{II}, sets $\rho^{\mathrm{II}}_{i^*}$ to be $3r^\star$, and adds (i^*, r^*) to $\mathcal{B}_0$. Then the algorithm iterates until $\Delta_{\mathcal{B}^\star}(j) < 5$ holds for all $j \in D$.

The following two observations show that this algorithm is well-defined. First, Observation 3 shows that the set $\bar{\mathcal{B}}$ at Step 3 of Algorithm 1 is always nonempty.

▶ **Observation 3.** *For all $j \in D$, there exists some $(i,r) \in \mathcal{B}_0$ such that $j \in B(i,r)$.*

Proof. From the feasibility of $(x^\star, y^\star, z^\star)$, there exists some $i \in F_\mu$ such that $z^\star_{i,j} > 0$, and this in turn implies that there exists some $r \in R_i$ such that $j \in B(i,r)$ and $x^\star_{i,r} > 0$. ◀

The following observation shows that ρ^{II} is unambiguously defined by Algorithm 1.

▶ **Observation 4.** *Step 4 of Algorithm 1 never chooses the same facility more than once.*

Proof. Suppose towards contradiction that Step 4 chooses (i, r_1) at some point and (i, r_2) at a later point during the execution of Algorithm 1 for some $r_1 \neq r_2$.

Suppose $r_1 < r_2$. Consider the moment the algorithm chooses (i, r_1). This implies that there exists some $j \in B(i, r_1)$ such that $\Delta_{\mathcal{B}^\star}(j) \geq 5$; since $B(i,r_1) \subseteq B(i,r_2)$, this implies $(i, r_2) \in \bar{\mathcal{B}}$, a contradiction to the design of the algorithm.

Suppose $r_1 > r_2$. Consider the moment the algorithm chooses (i, r_2). Since $B(i,r_2) \subseteq B(i,r_1)$ and $(i,r_1) \in \mathcal{B}^\star$, we have $\Delta_{\mathcal{B}^\star}(j) \leq 1$. Hence $(i,r_2) \notin \bar{\mathcal{B}}$ and cannot be picked in Step 4. ◀

The following lemma summarizes one of the key properties our algorithm aims to have.

▶ **Lemma 5.** *For any $(i^\star, r^\star) \in \mathcal{B}^\star$ and any $(i', r') \in \mathcal{B}_0$ such that $\pi_{i^\star}$ and (i', r') are adjacent in $\mathcal{G}$, we have $r' \leq r^\star$.*

Proof. Consider the moment the algorithm chooses $(i^\star, r^\star)$. Step 5 of the algorithm guarantees that $\Delta_{\mathcal{B}^\star}(\pi_{i^\star}) \geq 5$ and therefore $(i', r') \in \bar{\mathcal{B}}$. Since the algorithm chose $(i^\star, r^\star)$ over (i', r'), it shows that $r' \leq r^\star$. ◀

Connecting the opened facilities

To obtain the second solution $(S^{\mathrm{II}}, \rho^{\mathrm{II}}, T^{\mathrm{II}})$, it remains to build the Steiner tree T^{II}. Consider the following LP relaxation for the Steiner tree problem with vertex set $V := F \cup D$, edge set E, terminal set $W \subseteq V$, and a given root $t \in W$.

$$\begin{aligned} \text{minimize } & \sum_{e \in E} d_e \cdot h_e \\ \text{subject to } & \sum_{e \in \delta(U)} h_e \geq 1, \qquad \forall U \subseteq V \setminus \{t\} \text{ with } U \cap W \neq \emptyset, \\ & h \geq 0. \end{aligned} \tag{2}$$

Note that the costs of the edges in this relaxation are defined by d_e. In the following we construct a feasible solution for the above LP relaxation, where the set of terminals W is chosen as the set of proxy clients $\{\pi_i \mid i \in S^{\mathrm{II}}\}$.

We can assume without loss of generality on the variables $z^\star$ that, for all $j \in D$, $\sum_{i \in F_\mu} z^\star_{i,j} = 1$ for otherwise we can scale down $z^\star_{i,j}$ for all $i \in F_\mu$ simultaneously to make it so without losing the feasibility of the resulting solution. Construct a vector $p \in \mathbb{R}^E$ by setting

$$p_{(\pi_i, i')} := \begin{cases} z^\star_{i', \pi_i}, & \text{for all } i \in S^{\mathrm{II}} \text{ and } i' \in F_\mu, \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, in the above construction we fractionally wire π_i for each $i \in S^{\mathrm{II}}$ to all the facilities that fractionally covers π_i in $z^\star$. Since all the facilities are fractionally connected to the sink t in $y^\star$ by the LP constraint (1), it follows by the above construction that $y^\star + p$ fractionally connects the representative proxy client π_i to t for all $i \in S^{\mathrm{II}}$. Hence $y^\star + p$ is a feasible solution to (2).

Although we can use any LP-based algorithm for the Steiner tree problem at this point, let us assume that we use the LP-rounding algorithm of Jain [18] on this solution to construct a Steiner tree T_{pre} for the set of representative proxy clients in $W := \{\pi_i \mid i \in S^{\mathrm{II}}\}$. To obtain the desired Steiner tree T^{II}, we add edges (i, π_i) for all $i \in S^{\mathrm{II}}$ to T_{pre}.

The following lemma, which formally verifies the feasibility of $y^\star + p$ for (2), shows that the algorithm for this part is also well-defined, and a valid Steiner tree for W is produced.

▶ **Lemma 6.** *$y^\star + p$ is a feasible with respect to the constraint* (2)*.*

Proof. Consider an arbitrary terminal $\pi_{i^\star} \in W$ and an arbitrary set $U \subseteq V \setminus \{t\}$ such that $\pi_{i^\star} \in U$. We have

$$\begin{aligned} \sum_{e \in \delta(U)} (y^\star_e + p_e) & \geq \sum_{e \in \delta(U)} y^\star_e + \sum_{i' \in F_\mu \setminus U} p_{(\pi_{i^\star}, i')} \\ & \geq \sum_{i' \in F_\mu \cap U} z^\star_{i', \pi_{i^\star}} + \sum_{i' \in F_\mu \setminus U} z^\star_{i', \pi_{i^\star}} = \sum_{i' \in F_\mu} z^\star_{i', \pi_{i^\star}} = 1, \end{aligned}$$

where the second inequality follows from the feasibility of $y^\star$ and the construction of p and the last equality follows from the construction of the above algorithm. ◀

4 Analysis

In this section, we show that our algorithm is an approximation algorithm for the connected minimum sum of radii problem and establish the approximation guarantee.

Feasibility of the solutions

Consider each guess $t \in F$. It is clear that $(S_t^{\mathrm{I}}, \rho_t^{\mathrm{I}}, T_t^{\mathrm{I}})$ is a feasible solution. In the following we show that $(S_t^{\mathrm{II}}, \rho_t^{\mathrm{II}}, T_t^{\mathrm{II}})$ is also feasible.

By Lemma 6 and the correctness of Jain's rounding algorithm [18], T_t^{II} is indeed a Steiner tree for S_t^{II}. Hence, it suffices to prove the following lemma, which implies that, for all $j \in D$, there always exists some opened facility $i \in S^{\mathrm{II}}$ such that $d(i,j) \le \rho_i^{\mathrm{II}}$.

▶ **Lemma 7.** *For all $j \in D$, there exists some $(i^\star, r^\star) \in \mathcal{B}^\star$ such that $d(i^\star, j) \le 3r^\star$.*

Proof. Note that we have $\Delta_{\mathcal{B}^\star}(j) = +\infty$ at the beginning and $\Delta_{\mathcal{B}^\star}(j) < 5$ at the end of the execution of Algorithm 1. Consider the iteration at which $\Delta_{\mathcal{B}^\star}(j)$ becomes smaller than 5 for the first time and let $(i^\star, r^\star)$ be the ball chosen at Step 4 during this iteration. Since $\mathcal{G}$ is bipartite, $\Delta_{\mathcal{B}^\star}(j)$ becomes 1 or 3 at this iteration. If it becomes 1, this implies $j \in B(i^\star, r^\star)$ and there is nothing to prove. If $\Delta_{\mathcal{B}^\star}(j)$ becomes 3, this implies that there exists a path of length three between $(i^\star, r^\star)$ and j in $\mathcal{G}$; let $(i^\star, r^\star) - j' - (i', r') - j$ denote this path. At the beginning of this iteration, $\Delta_{\mathcal{B}^\star}(j)$ was no smaller than 5 and therefore $(i', r') \in \bar{\mathcal{B}}$. Since the algorithm chose $(i^\star, r^\star)$ over (i', r'), we have $r^\star \ge r'$, yielding $d(i^\star, j) \le d(i^\star, j') + d(j', i') + d(i', j) \le r^\star + r' + r' \le 3r^\star$. ◀

Approximation Guarantee

In the following we establish the approximation guarantee. Let $(S^{\mathsf{opt}}, r^{\mathsf{opt}}, T^{\mathsf{opt}})$ be an optimal solution and OPT denote its cost.

If $|S^{\mathsf{opt}}| = 1$, then the facility in S^{opt} will be iterated by the algorithm. Denote this facility by t^*. Then $(S_{t^*}^{\mathrm{I}}, \rho_{t^*}^{\mathrm{I}}, T_{t^*}^{\mathrm{I}})$ is an optimal solution and there is nothing to prove.

In the following we assume that $|S^{\mathsf{opt}}| \ge 2$. Since the algorithm iterates over all possible guesses, we assume without loss of generality that t is the facility with the smallest m_t value in S^{opt}, i.e.,

$$t \in S^{\mathsf{opt}} \quad \text{and} \quad t = \mathrm{argmin}_{i \in S^{\mathsf{opt}}} m_i.$$

Depending on whether or not $t \in F_\mu$, we further consider two cases. The following lemma shows that $(S_t^{\mathrm{I}}, \rho_t^{\mathrm{I}}, T_t^{\mathrm{I}})$ is a μ-approximation solution if $t \notin F_\mu$.

▶ **Lemma 8.** *If $t \in S^{\mathsf{opt}}$ and $t \notin F_\mu$, then*

$$OPT \;\ge\; \frac{1}{\mu} \cdot m_t \cdot \max_{j \in D} d(t, j).$$

Proof. We have $m_t < \mu$ by the assumption. Let $j^o := \arg\max_{j \in D} d(t, j)$ be the client that defines the radius ρ_t^{I}. Let i' be a facility in S^{opt} with $d(i', j^o) \le \rho_{i'}^{\mathsf{opt}}$. We have

$$\begin{aligned}
\mathsf{OPT} &= \sum_{i\in S^{\mathsf{opt}}} m_i\cdot\rho_i^{\mathsf{opt}} \;+\; \sum_{e\in T^{\mathsf{opt}}} d_e\\
&\ge m_{i'}\cdot\rho_{i'}^{\mathsf{opt}} \;+\; d(t,i')\\
&\ge m_t\cdot d(i',j^o) \;+\; \frac{1}{\mu}m_t\cdot d(t,i')\\
&\ge \frac{1}{\mu}\cdot m_t\cdot d(t,j^o) \;\;=\;\; \frac{1}{\mu}\cdot m_t\cdot \max_{j\in D} d(t,j),
\end{aligned}$$

where in the last inequality we apply the triangle inequality and the fact that $\mu\ge 1$. ◀

It remains to consider the case that $t\in F_\mu$, which in particular implies that $S^{\mathsf{opt}}\subseteq F_\mu$. We prove in the following that $(S_t^{\mathrm{II}},\rho_t^{\mathrm{II}},T_t^{\mathrm{II}})$ is a $(5+\frac{3}{\mu})$-approximation solution in this case.

Since $S^{\mathsf{opt}}\subseteq F_\mu$, it follows that the LP (1) admits $(S^{\mathsf{opt}},r^{\mathsf{opt}},T^{\mathsf{opt}})$ as a feasible solution. Hence the cost of the fractional solution $(x^\star,y^\star,z^\star)$ provides a lower-bound for OPT. Similarly to the facility location problem [4] and the minimum sum of radii problem [9], we use the dual optimal solution to bound the cost of the rounded solution via complementary slackness. Consider the dual LP of the LP (1), which we provide below, and let $(\alpha^\star,\beta^\star,\gamma^\star,\lambda^\star)$ be an optimal solution for it.

$$\begin{aligned}
\text{maximize}\quad & \sum_{j\in D}\alpha_j\\
\text{subject to}\quad & \sum_{j\in B(i,r)}\gamma_{i,j}\;\le\; m_i\cdot r, && \forall i\in F_\mu, r\in R_i,\\
& \alpha_j-\sum_{U\subseteq V\setminus\{t\}:i\in U}\beta_{j,U}\;\le\;\gamma_{i,j}, && \forall i\in F_\mu, j\in D,\\
& \sum_{j\in D}\sum_{U\subseteq V\setminus\{t\}:e\in\delta(U)}\beta_{j,U}\;\le\; d_e, && \forall e\in E,\\
& \alpha,\beta,\gamma\ge 0.
\end{aligned}\tag{3}$$

The following lemma bounds the weighted cost of a facility in term of the dual values of the clients contained within. Intuitively, it follows from standard complementary slackness conditions between $(x^\star,y^\star,z^\star)$ and $(\alpha^\star,\beta^\star,\gamma^\star,\lambda^\star)$.

▶ **Lemma 9.** *For any $i\in F_\mu$ and any $r\in R_i$, we have $x^\star_{i,r}>0$ implies that $m_i\cdot r\le\sum_{j\in B(i,r)}\alpha^\star_j$.*

Proof. From the complementary slackness condition, $x^\star_{i,r}>0$ implies

$$\sum_{j\in B(i,r)}\gamma^\star_{i,j}\;=\;m_i\cdot r.\tag{4}$$

Consider an arbitrary $j\in B(i,r)$. If $\gamma^\star_{i,j}>0$, we have from the complementary slackness that

$$z^\star_{i,j}\;=\;\sum_{r'\in R_i:j\in B(i,r')}x^\star_{i,r'}\;>\;x^\star_{i,r}\;>\;0.$$

By complementary slackness condition again this implies

$$\gamma^\star_{i,j} \;=\; \alpha^\star_j - \sum_{U \subseteq V \setminus \{t\} : i \in U} \beta^\star_{j,U} \;\leq\; \alpha^\star_j. \tag{5}$$

On the other hand, if $\gamma^\star_{i,j} = 0$, it trivially holds that

$$\gamma^\star_{i,j} \leq \alpha^\star_j. \tag{6}$$

Combining (5) and (6) with (4) yields $m_i \cdot r \;\leq\; \sum_{j \in B(i,r)} \alpha^\star_j$. ◀

Consider any $(i_1, r_1), (i_2, r_2) \in \mathcal{B}^\star$ such that $i_1 \neq i_2$. By the design of the rounding procedure in the first part of the algorithm, we always have that $B(i_1, r_1)$ and $B(i_2, r_2)$ are disjoint. Hence, combining this fact with Lemma 9, the total weighted facility cost can be bounded as

$$\begin{aligned}\sum_{i \in S^{\mathrm{II}}_t} m_i \cdot \rho^{\mathrm{II}}_i \;&=\; \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} 3 \cdot m_i \cdot r^\star \\ &\leq\; \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} \left(3 \cdot \sum_{j \in B(i^\star, r^\star)} \alpha^\star_j \right) \;=\; 3 \cdot \sum_{j \in D} \alpha^\star_j \;\leq\; 3 \cdot \mathsf{OPT}.\end{aligned} \tag{7}$$

In the following we consider the cost incurred by the Steiner tree T^{II}_t. We have the following lemma regarding the value of the solution $y^\star + p$ with respect to LP (2).

▶ **Lemma 10.**

$$\sum_{e \in E} d_e \cdot (y^\star_e + p_e) \;\leq\; \left(1 + \frac{1}{\mu}\right) \cdot \mathsf{OPT}.$$

Proof. By the construction of p we have

$$\sum_{e \in E} d_e \cdot y^\star_e + \sum_{e \in E} d_e \cdot p_e \;\leq\; \mathsf{OPT} \;+\; \sum_{i^\star \in S^{\mathrm{II}}} \sum_{i' \in F_\mu} d(i', \pi_{i^\star}) \cdot z^\star_{i', \pi_{i^\star}}. \tag{8}$$

For any (i, j) such that $z^\star_{i,j} > 0$, the feasibility of $(x^\star, y^\star, z^\star)$ implies that there must exist some $r \in R_i$ such that $j \in B(i, r)$ and $x^\star_{i,r} > 0$. This yields

$$\begin{aligned}\sum_{i^\star \in S^{\mathrm{II}}} \sum_{i' \in F_\mu} d(i', \pi_{i^\star}) \cdot z^\star_{i', \pi_{i^\star}} \;&=\; \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} \sum_{i' \in F_\mu} d(i', \pi_{i^\star}) \cdot z^\star_{i', \pi_{i^\star}} \\ &\leq\; \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} \sum_{i' \in F_\mu} r^\star \cdot z^\star_{i', \pi_{i^\star}} \\ &=\; \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} r^\star \;\leq\; \sum_{i^\star \in S^{\mathrm{II}}} \frac{m_{i^\star}}{\mu} \cdot r^\star \;\leq\; \frac{1}{\mu} \cdot \mathsf{OPT},\end{aligned}$$

where the first inequality follows from Lemma 5 and the fact that $z^\star_{i' \pi_{i^\star}} > 0$ implies that there exists some $r' \in R_{i'}$ such that $\pi_{i^\star} \in B(i', r')$ and $x^\star_{i', r'} > 0$, the second equality follows from the construction in the second part of the algorithm, the second inequality from $S^{\mathrm{II}} \subseteq F_\mu$ which implies that $m_i \geq \mu$ for all $i \in F_\mu$, and the last inequality follows from (7). ◀

By the design of the algorithm for constructing T_t^{II}, the bound in Lemma 10, and the fact that Jain's rounding algorithm gives a 2-approximation [18], we have

$$\begin{aligned}
\sum_{e \in T_t^{\mathrm{II}}} d_e &= \sum_{e \in T_{\mathsf{pre}}} d_e + \sum_{i^\star \in S_t^{\mathrm{II}}} d(i^\star, \pi_{i^\star}) \\
&\le \left(2 + \frac{2}{\mu}\right) \cdot \mathsf{OPT} + \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} r^\star \\
&\le \left(2 + \frac{2}{\mu}\right) \cdot \mathsf{OPT} + \sum_{(i^\star, r^\star) \in \mathcal{B}^\star} \frac{m_{i^\star}}{\mu} \cdot r^\star \le \left(2 + \frac{3}{\mu}\right) \cdot \mathsf{OPT}, \qquad (9)
\end{aligned}$$

where in the second last inequality we use the fact that $m_{i^\star} \ge \mu$ for all $i^\star \in S_t^{\mathrm{II}}$ and in last inequality we apply Inequality (7). Combining Inequalities (7) and (9), we obtain

$$\sum_{i \in S_t^{\mathrm{II}}} m_i \cdot \rho_i^{\mathrm{II}} + \sum_{e \in T_t^{\mathrm{II}}} d_e \le \left(5 + \frac{3}{\mu}\right) \cdot \mathsf{OPT}.$$

This proves the following theorem. Choosing $\mu := \frac{5+\sqrt{37}}{2} < 5.542$ yields a μ-approximation algorithm.

▶ **Theorem 11.** *The given algorithm is a* $\max\left(\mu, 5 + \frac{3}{\mu}\right)$*-approximation algorithm.*

5 NP-hardness Results

In this section, we prove Theorem 2 by showing that the problem remains NP-hard even for two special cases. First, the following theorem shows that this problem remains NP-hard even when we only allow clusters with zero radii.

▶ **Theorem 12.** *The* connected minimum sum of radii *problem is NP-hard when* $m_i = +\infty$ *for all* $i \in F$ *and* $M = 1$*.*

Proof. We give a reduction from the Metric Steiner Tree problem, which is known to be NP-complete [20]. Construct an instance of the connected minimum sum of radii problem where the terminals in the Steiner tree instance become facilities and clients at the same time. Observe that an optimal solution to this instance opens all terminals, set their radii to zeroes, and takes a Steiner tree connecting them. ◀

On the other hand, the following theorem shows that the NP-hardness remain true even when no connection between opened facilities is required. The proof closely follows the NP-hardness proof of the (non-connected) minimum sum of radii problem [11]; but we present the full proof here for the sake of completeness.

▶ **Theorem 13.** *The* connected minimum sum of radii *problem is NP-hard when* $m_i = 1$ *for all* $i \in F$ *and* $M = 0$*.*

Proof. We give a reduction from 3SAT [5]. Consider an instance of 3SAT with n variables $x_1, \ldots, x_n$ and k clauses $C_1, \ldots, C_k$. We construct an instance of the connected minimum sum of radii problem as follows.

Let $F := \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$ and $D := \{C_1, \ldots, C_k, v_1, \ldots, v_n\}$. To define a metric on $V := F \cup D$, consider a weighted graph on the vertex set V, where we have an edge (x_i, C_j) (or $(\bar{x}_i, C_j)$, respectively) of weight 2^{i-1} if and only if C_j contains x_i (or $\bar{x}_i$). We also add edges (x_i, v_i) and $(\bar{x}_i, v_i)$ of weight 2^{i-1} for all $i = 1, \ldots, n$. The metric d is then defined as the shortest path metric on this weighted graph. We claim that the optimal solution value to this constructed instance is at most $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$ if and only if the 3SAT instance is satisfiable.

Suppose that the 3SAT instance is satisfiable. Fix a satisfying assignment. For each variable x_i, we open x_i (or $\bar{x}_i$, respectively) if x_i is true (or false) under the fixed assignment and set its radius to 2^{i-1}. This yields a solution of value $2^n - 1$ in which every client can be assigned.

Conversely, suppose that there exists a solution to the constructed instance of the connected minimum sum of radii problem whose value is at most $2^n - 1$. Fix such a solution. Suppose towards contradiction that there exists some k such that neither x_k nor $\bar{x}_k$ is open with radius at least 2^{k-1}. Let $k^\star$ be the largest such k. Then there must exist some ℓ such that $v_{k^\star}$ is assigned to x_ℓ or $\bar{x}_\ell$. Note that $d(v_{k^\star}, x_\ell) = d(v_{k^\star}, \bar{x}_\ell) \geq 2 \cdot 2^{k^\star-1} + 2^{\ell-1}$ since every edge incident with $x_{k^\star}$ or $\bar{x}_{k^\star}$ is of weight $2^{k^\star-1}$ and every edge incident with x_ℓ or $\bar{x}_\ell$ is of weight $2^{\ell-1}$. If $\ell > k^\star$, the total cost of the solution must be at least

$$\sum_{i \in \{k^\star+1,\ldots,n\} \setminus \{\ell\}} 2^{i-1} + (2 \cdot 2^{k^\star-1} + 2^{\ell-1}) > 2^n - 1,$$

which leads to contradiction. If $\ell < k^\star$, the total cost of the solution must be at least

$$\sum_{i \in \{k^\star+1,\ldots,n\}} 2^{i-1} + (2 \cdot 2^{k^\star-1} + 2^{\ell-1}) > 2^n - 1,$$

leading to contradiction again.

We thus have that, for all $i = 1, \ldots, n$, x_i or $\bar{x}_i$ (or both) is open with radius at least 2^{i-1}. Since $2^n - 1 = \sum_{i=1}^{n} 2^{i-1}$, this implies that exactly one of x_i and $\bar{x}_i$ is open with radius exactly 2^{i-1} for all $i = 1, \ldots, n$. (Note that opening with zero radius is useless.) Consider a truth value assignment that sets x_i to true if x_i is open, and false otherwise. Observe that this is a satisfying assignment. ◀

References

1 Matthew Andrews and Lisa Zhang. The access network design problem. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 40–59. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743427.

2 Moritz Buchem, Katja Ettmayr, Hugo K. K. Rosado, and Andreas Wiese. A $(3+\varepsilon)$-approximation algorithm for the minimum sum of radii problem with outliers and extensions for generalized lower bounds. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1738–1765. SIAM, 2024. doi:10.1137/1.9781611977912.69.

3 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 1–10. ACM, 2001. doi:10.1145/380752.380753.

4 Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003. doi:10.1137/S0097539703405754.

5 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

6 Srinivas Doddi, Madhav V. Marathe, S. S. Ravi, David Scot Taylor, and Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2000. doi:10.1007/3-540-44985-X_22.

7 Ding-Zhu Du and Peng-Jun Wan. *Connected Dominating Set: Theory and Applications*. Springer Publishing Company, Incorporated, 2012.

8 Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Approximating connected facility location problems via random facility sampling and core detouring. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1174–1183. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347210.

9 Zachary Friggstad and Mahya Jamshidian. Improved polynomial-time approximations for clustering with minimum sum of radii or diameters. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 56:1–56:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.56.

10 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximating connected facility location with lower and upper bounds via LP rounding. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 1:1–1:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.SWAT.2016.1.

11 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, 2010. doi:10.1007/S00453-009-9282-7.

12 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

13 Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998. doi:10.1007/PL00009201.

14 Anupam Gupta, Jon M. Kleinberg, Amit Kumar, Rajeev Rastogi, and Bülent Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 389–398. ACM, 2001. doi:10.1145/380752.380830.

15 Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 365–372. ACM, 2003. doi:10.1145/780542.780597.

16 Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Math. Program.*, 79:191–215, 1997. doi:10.1007/BF02614317.

17 D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10:180–184, 1985. doi:10.1287/MOOR.10.2.180.

18 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Comb.*, 21(1):39–60, 2001. doi:10.1007/S004930170004.

19 Riheng Jia, Jinhao Wu, Jianfeng Lu, Minglu Li, Feilong Lin, and Zhonglong Zheng. Energy saving in heterogeneous wireless rechargeable sensor networks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, May 2-5, 2022*, pages 1838–1847. IEEE, 2022. doi:10.1109/INFOCOM48880.2022.9796770.

20 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

21 Samir Khuller, Manish Purohit, and Kanthi K. Sarpatwar. Analyzing the optimal neighborhood: Algorithms for budgeted and partial connected dominating set problems. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1702–1713. SIAM, 2014. doi:10.1137/1.9781611973402.123.

22 Samir Khuller and Sheng Yang. Revisiting connected dominating sets: An almost optimal local information algorithm. *Algorithmica*, 81(6):2592–2605, 2019. doi:10.1007/S00453-019-00545-0.

23 Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004. doi:10.1007/S00453-004-1112-3.

24 Wenzheng Xu, Weifa Liang, Xiaohua Jia, Haibin Kan, Yinlong Xu, and Xinming Zhang. Minimizing the maximum charging delay of multiple mobile chargers under the multi-node energy charging scheme. *IEEE Trans. Mob. Comput.*, 20(5):1846–1861, 2021. doi:10.1109/TMC.2020.2973979.

Lower Bounds for Adaptive Relaxation-Based Algorithms for Single-Source Shortest Paths

Sunny Atalig
University of California, Riverside, CA, USA

Alexander Hickerson
University of California, Riverside, CA, USA

Arrdya Srivastav
University of California, Riverside, CA, USA

Tingting Zheng
Guangdong University of Technology, Guangzhou, China

Marek Chrobak
University of California, Riverside, CA, USA

Abstract

We consider the classical single-source shortest path problem in directed weighted graphs. D. Eppstein proved recently an $\Omega(n^3)$ lower bound for oblivious algorithms that use relaxation operations to update the tentative distances from the source vertex. We generalize this result by extending this $\Omega(n^3)$ lower bound to *adaptive* algorithms that, in addition to relaxations, can perform queries involving some simple types of linear inequalities between edge weights and tentative distances. Our model captures as a special case the operations on tentative distances used by Dijkstra's algorithm.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases single-source shortest paths, lower bounds, decision trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.8

Funding Research partially supported by National Science Foundation grant CCF-2153723.

1 Introduction

We consider the classical single-source shortest path problem in directed weighted graphs. In the case when all edge weights are non-negative, Dijkstra's algorithm [8], if implemented using Fibonacci heaps, computes the shortest paths in time $O(m + n \log n)$, where n is the number of vertices and m is the number of edges. In the general case, when negative weights are allowed (but not negative cycles), the Bellman-Ford algorithm [20, 2, 19, 12] solves this problem in time $O(nm)$.

Both algorithms work by repeatedly executing operations of *relaxations*. (This type of algorithms are also sometimes called label-setting algorithms [7].) Let ℓ_{uv} denote the weight of an edge (u, v). For each vertex v, these algorithms maintain a value $D[v]$ (that we will refer to as the *D-value* at v) that represents the current upper bound on the distance from the source vertex s to v. A relaxation operation for an edge (u, v) replaces $D[v]$ by $\min\{D[v], D[u] + \ell_{uv}\}$. That is, $D[v]$ is replaced by $D[u] + \ell_{uv}$ if visiting v via u turns out to give a shorter distance to v, based on the current distance estimates. When the algorithm completes, each value $D[v]$ is equal to the correct distance from s to v. Dijkstra's algorithm executes only one relaxation for each edge, while in the Bellman-Ford algorithm each edge can be relaxed $\Theta(n)$ times.

We focus on the case of complete directed graphs, in which case $m = n(n-1)$. For complete graphs, the number of relaxations in Dijkstra's algorithm is $\Theta(n^2)$. In contrast, the Bellman-Ford algorithm executes $\Theta(n^3)$ relaxations. This raises the following natural question: *is it possible to solve the shortest-path problem by using asymptotically fewer than $O(n^3)$ relaxations, even if negative weights are allowed?*

To make this question meaningful, some restrictions need to be imposed on allowed algorithms. Otherwise, an algorithm can "cheat": it can compute the shortest paths without any explicit use of relaxations, and then execute $n-1$ relaxations on the edges in the shortest-path tree, in order of their hop-distance from s, thus making only $n-1$ relaxations.

Eppstein [9] circumvented this issue by assuming a model where the sequence of relaxations is independent of the weight assignment. Then the question is whether there is a short "universal" sequence of relaxations, namely one that works for an arbitrary weight assignment. The Bellman-Ford algorithm is essentially such a universal sequence of length $O(n^3)$. Eppstein [9] proved that this is asymptotically best possible; that is, each universal relaxation sequence must have $\Omega(n^3)$ relaxations. This lower bound applies even in the randomized case, when the relaxation sequence is generated randomly and the objective is to minimize the expected number of relaxations.

The question left open in [9] is whether the $\Omega(n^3)$ lower bound applies to relaxation-based *adaptive* algorithms, that generate relaxations based on information collected during the computation. (This problem is also mentioned by Hu and Kozma [15], who remark that lower bounds for adaptive algorithms have been "elusive".) We answer this question in the affirmative for some natural types of adaptive algorithms.

In our computation model, an algorithm is allowed to perform two types of operations: (i) *queries*, which are simple linear inequalities involving edge weights and D-values, and (ii) *relaxation updates*, that modify D-values. The action at each step depends on the outcomes of the earlier executed queries. Such algorithms can be represented as decision trees, with queries and updates in their nodes, and with each query node having two children, one corresponding to the "yes" outcome and the other to the "no" outcome.

Specifically, we study *query/relaxation-based algorithms* that can make queries of three types:

D-comparison query: "$D[u] < D[v]$?", for two vertices u, v,
Weight-comparison query: "$\ell_{uv} < \ell_{xy}$?", for two edges (u,v), (x,y),
Edge query: "$D[u] + \ell_{uv} < D[v]$?", for an edge (u,v),
and can update D-values as follows:
Relaxation update: "$D[v] \leftarrow \min\{D[v], D[u] + \ell_{uv}\}$", for an edge (u,v).

Throughout the paper, for brevity, we will write "D-query" instead of "D-comparison query" and "weight query" instead of "weight-comparison query".

We assume that initially $D[s] = 0$ and $D[v] = \ell_{sv}$ for all vertices $v \neq s$. This initialization and the form of relaxation updates ensure that at all times each value $D[v]$ represents the length of some simple path from s to v. Thus D-queries and edge queries amount to comparing the lengths of two paths from s. Further, the D-values induce a tentative approximation of the shortest-path tree, where a node u is the parent of a node v if the last decrease of $D[v]$ resulted from a relaxation of edge (u,v). So the algorithm's decision at each step depends on this tentative shortest-path tree.

Our contributions. We start by considering algorithms that use only edge queries. For such algorithms we prove the following $\Omega(n^3)$ lower bound:

▶ **Theorem 1.** *(a) Let $\mathcal{A}$ be a deterministic query/relaxation-based algorithm for the single-source shortest path problem that uses only edge queries. Then the running time of $\mathcal{A}$ is $\Omega(n^3)$, even if the weights are non-negative and symmetric (that is, the graph is undirected). (b) If $\mathcal{A}$ is a randomized algorithm then the same $\Omega(n^3)$ lower bound holds for $\mathcal{A}$'s expected running time.*

We first give the proof of Theorem 1(a), the lower bound for deterministic algorithms. In this proof, (in Section 3), we view the computation of $\mathcal{A}$ as a game against an adversary who gradually constructs a weight assignment, consistent with the queries, on which most of the edge queries performed by $\mathcal{A}$ will have negative outcomes, thus revealing little information to $\mathcal{A}$ about the structure of the shortest-path tree.

Then, in Section 4, we show how to extend this lower bound to all three types of queries if negative weights are allowed, proving Theorem 2(a) below.

▶ **Theorem 2.** *(a) Let $\mathcal{A}$ be a deterministic query/relaxation algorithm for the single-source shortest path problem that uses the three types of queries: D-queries, weight-queries, and edge queries, as well as relaxation updates. Then the running time of $\mathcal{A}$ is $\Omega(n^3)$. (b) If $\mathcal{A}$ is a randomized algorithm then the same $\Omega(n^3)$ lower bound holds for $\mathcal{A}$'s expected running time.*

Our query/relaxation model captures as a special case the operations on tentative distances used by Dijkstra's algorithm, because D-queries are sufficient to maintain the ordering of vertices according to their D-values. More broadly, Theorem 2 may be helpful in guiding future research on speeding up shortest-path algorithms for the general case, when negative weights are allowed, by showing limitations of naïve approaches based on extending Dijkstra's algorithm.

The proof of Theorem 2(a) is essentially via a reduction, showing that the model with all three types of queries can be reduced to the one with only edge queries, and then applying the lower bound from Theorem 1(a). This reduction modifies the weight assignment, making it asymmetric and introducing negative weights. As a side result, we also observe in Theorem 4 that this reduction works even for arbitrary (not necessarily complete) graphs, giving a lower bound that generalizes the one in [9], as it applies to adaptive algorithms in our query/relaxation model.

Finally, in Section 5 we extend both lower bounds to randomized algorithms. The proofs are based on Yao's principle [22]; that is, we give a probability distribution on weight assignments on which any deterministic algorithm performs poorly.

Our lower bound results are valid even if all weights are integers of polynomial size. In the proof of Theorem 1 all weights are non-negative integers with maximum value $\ell_{\max} = O(n)$. The proof of Theorem 2 uses Golomb rulers [21, 10, 4] (also known as Sidon sets) to construct weight assignments with maximum value $\ell_{\max} = O(n^4)$. In the randomized case, these bounds increase by a factor of $O(n)$.

As explained near the end of Section 5, the lower bounds for expectation in Theorems 1(b) and 2(b) can be quite easily extended to high-probability bounds.

Related work. As earlier mentioned, the Bellman-Ford algorithm can be thought of as a universal relaxation sequence. It consists of $n-1$ iterations with each iteration relaxing all edges in some pre-determined order, so the length of this sequence is $(1+o(1))n^3$. The leading constant 1 in this bound was reduced to $\frac{1}{2}$ by Yen [23], who designed a universal sequence with $(\frac{1}{2}+o(1))n^3$ relaxations. Eppstein's lower bound in [9] shows in fact a lower bound of $\frac{1}{6}$ on the leading constant, and just recently Hu and Kozma [15] proved that constant $\frac{1}{2}$ is in fact optimal.

Bannister and Eppstein [1] showed that the leading constant can be reduced to $\frac{1}{3}$ with randomization, namely that there is a probability distribution on relaxation sequences for which a sequence, drawn from this distribution, will compute correct distances in expected time $(\frac{1}{3} + o(1))n^3$ (or even with high probability). Eppstein's lower bound proof [9] for randomized sequences shows that this constant is at least $\frac{1}{12}$.

Some of the above-mentioned papers extend the results to graphs that are not necessarily complete. In particular, Eppstein [9] proved that for n-vertex graphs with m edges, $\Omega(mn/\log n)$ relaxations are necessary.

The average-case complexity of the Bellman-Ford and Dijkstra's algorithms has also been studied. For example, Meyer et al. [18] show that the Bellman-Ford algorithm requires $\Omega(n^2)$ steps on average, if the weights are uniformly distributed random numbers from interval $[0, 1]$.

Some work has been done on improving lower and upper bounds in models beyond our query/relaxation setting. Of those, the recent breakthrough paper by Fineman [11] is particularly relevant. It gives a randomized $\tilde{O}(mn^{8/9})$-expected-time algorithm for computing single-source shortest paths with arbitrary weights. Fineman's computation model is not far from ours in the sense that the weights are arbitrary real numbers and the only arithmetic operations on weights are additions and subtractions, but it also needs branch instructions that cannot be expressed using our queries.

The special case when weights are integers is natural and has been extensively investigated (see [6, 13, 3], for example). In the integer domain one can extract information about the weight distribution, and thus about the structure of the shortest-path tree, using operations other than linear inequalities involving weights. The state-of-the-art in this model is the (randomized) algorithm by Bernstein et al. [3] that achieves running time $O(m \log^8 n \log W)$ with high probability for weight assignments where the smallest weight is at least $-W$ (and $W \geq 2$).

Some lower bounds have also been reported for related problems, for example for shortest paths with restrictions on the number of hops [5, 14, 17] or k-walks [16].

2 Preliminaries

The input is a weighted complete directed graph G. The set of all vertices of G is denoted by V, and $s \in V$ is designated as the *start vertex*. The set of all edges of G is denoted by E and a *weight assignment* is a function $\ell \colon E \to \mathbb{Z}$. (While real-valued weights are common in the literature, in our constructions we only need integers.) We will use notations $\ell(u, v)$ and ℓ_{uv} for the weight of an edge (u, v). By $\ell_{\max}$ we denote the maximum absolute value of an edge weight, that is $\ell_{\max} = \max_{(u,v)\in E} |\ell_{uv}|$.

Whenever we write "path" we mean a "simple path", that is a path where each vertex is visited at most once. The *distance from x to y* is defined as the length of the shortest path from x to y. We will assume that the input graph does not have negative cycles. Note that this assumption gives an algorithm additional information that can potentially be used to reduce the running time.

The edges in the shortest paths from s to all other vertices form a tree that is called the shortest-path tree. The root of this tree is s. (There is a minor subtlety here related to ties. A more precise statement is that *there is a way to break ties*, so that the shortest paths form a tree.)

Formalizing query/relaxation models. We now formally define our computation model. We assume that each vertex v has an associated value $D[v]$, called the D-value at v. Initially $D[s] = 0$ and $D[v] = \ell_{sv}$ for $v \neq s$. A *query* is a boolean function whose arguments are edge weights and D-values. A *query model* $\mathbb{Q}$ is simply a set of allowed queries. For example, the model that has only edge queries is $\mathbb{Q} = \{1_{D[v]<D[u]+\ell_{uv}} \mid (u,v) \in E\}$, where 1_ξ is the indicator function for a predicate ξ. The query/relaxation model in Theorem 1 has query model $\mathbb{Q}$ consisting of all D-queries, weight-queries, and edge queries. (The reduction in Section 4 is actually for an even more general query model.)

An algorithm $\mathcal{A}$ using a query/relaxation model $\mathbb{Q}$ is then a decision tree, where each internal node corresponds to either a query from $\mathbb{Q}$ (with one "yes" and one "no" branch) or a relaxation operation (which has one branch), and each leaf is a relaxation operation. (These leaves have no special meaning.) With this definition, at any step of the computation, $D[v]$ represents the length of a path from s to v. This decision tree must correctly compute all distances from s; that is, for each weight assignment ℓ, when the computation of $\mathcal{A}$ reaches a leaf then for each vertex v the value of $D[v]$ must be equal to the distance from s to v.

The running time of $\mathcal{A}$ for a weight assignment ℓ is defined as the number of steps performed by $\mathcal{A}$ until each value $D[v]$ is equal to the correct distance from s to v. Notice that this is not the same as the depth of the decision tree, which could be greater. (This definition matches the concept of "reduced cost" used in [9] for non-adaptive algorithms. For deterministic algorithms we could as well define the running time as the maximum tree depth, but this definition wouldn't work in the randomized case.)

Edge weights using potential functions. We define a *potential function* as a function $\phi : V \to \mathbb{Z}$ with $\phi(s) = 0$. (These functions are also sometimes called *price functions* in the literature.) To reduce clutter, we will sometimes write the potential value on v as ϕ_v instead of $\phi(v)$.

A potential function induces a weight assignment $\Delta\phi$ defined by $\Delta\phi(u,v) = \phi_v - \phi_u$, for each $(u,v) \in E$. Such potential-induced weights satisfy the following *path independence property:* For any two vertices u, v, all paths from u to v have the same length, namely $\Delta\phi(u,v)$. Note that $\Delta\phi$ will have some negative weights, unless ϕ is identically 0, but it does not form negative cycles. Also, every spanning tree rooted at s is a shortest-path tree for $\Delta\phi$.

Any weight assignment ℓ can be combined with a potential function ϕ to obtain a new weight assignment $\ell' = \ell + \Delta\phi$. Such ℓ' satisfies the following *distance preservation property:* For any two vertices u, v and any path P from u to v, we have $\ell'(P) = \ell(P) + \phi_v - \phi_u$.

Due to the above properties, potential functions have played a key role in the most recent single-source shortest path algorithms [3, 11], in particular being used to transform a negative weight assignment into a non-negative one so that Dijkstra's algorithm can be applied. However, in this paper we will use them for an entirely different purpose, which is to construct difficult weight assignments in Section 4. Roughly, a potential-induced assignment $\Delta\phi$ can act as a "mask" on top of existing weights that renders D-queries and weight queries useless.

3 Lower Bound for Deterministic Algorithms with Edge Queries

This section gives the proof of Theorem 1(a). That is, we prove that every deterministic algorithm that uses relaxations and edge queries needs to make $\Omega(n^3)$ operations to compute correct distances. This lower bound applies even if all weights are non-negative and the weight assignment is symmetric. (One can think of it as an undirected graph, although we emphasize that in the proof below we use directed edges.)

For the proof, fix an algorithm $\mathcal{A}$. We will show how to construct a weight assignment such that only after $\Omega(n^3)$ operations the D-values computed by $\mathcal{A}$ represent the correct distances from the source vertex.

Each weight assignment considered in our construction is symmetric and is uniquely specified by a permutation of the vertices. The weight assignment corresponding to a permutation $\pi = x_0, x_1, ..., x_{n-1}$, where $x_0 = s$, is defined as follows: for any $0 \leq i < j < n$,

$$\ell_\pi(x_i, x_j) \;=\; \begin{cases} 2 & \text{if } j = i+1 \\ L - 5i/2 & \text{if } j \geq i+2 \text{ and } i \text{ is even} \\ L & \text{if } j \geq i+2 \text{ and } i \text{ is odd} \end{cases}$$

where L is some sufficiently large integer, say $L = 5n$. Then the shortest path tree is just a Hamiltonian path $x_0, x_1, ..., x_{n-1}$. Note that the distance between any two vertices is less than $2n$, while each edge not on this path has length larger than $2n$.

The proof is by showing an adversary strategy that gradually constructs a permutation of the vertices in response to $\mathcal{A}$'s operations. The strategy consists of $(n-1)/2$ phases. (For simplicity, assume that n is odd.) When a phase k starts, for $k = 1, ..., (n-1)/2$, the adversary will have already revealed a prefix $X_{k-1} = x_0, x_1, ..., x_{2k-2}$ of the final permutation. The goal of this phase is to extend X_{k-1} by two more vertices, responding to $\mathcal{A}$'s queries and updates so as to force $\mathcal{A}$ to make as many operations as possible within the phase, without revealing anything about the rest of the permutation.

To streamline the proof, we think about the initial state as following the non-existent $0'$th phase, and we assume that the D-values for all vertices other than s are initialized to $L+1$, instead of L.

We now describe the adversary strategy in phase k, by specifying how the adversary responds to each operation of $\mathcal{A}$ executed in this phase. Let $Y_{k-1} = V \setminus X_{k-1}$, let A be set of the edges from x_{2k-2} to Y_{k-1} and B be the set of edges inside Y_{k-1}. The adversary will maintain marks on the edges in $A \cup B$, starting with all edges unmarked. We will say that $\mathcal{A}$ *accesses* an edge (u, v) if it executes either an edge query or a relaxation for (u, v).

The idea is this: because of the choice of edge weights and the invariants on the D-values (to be presented soon), each edge query for an edge $(x_{2k-2}, y) \in A$ not yet relaxed in this phase will have a positive outcome. This way, these responses will not reveal what the next vertex x_{2k-1} on the path is. The adversary waits until $\mathcal{A}$ relaxes all these edges, and keeps track of these relaxations by marking the relaxed edges. At the same time, $\mathcal{A}$ may be accessing edges in B. The adversary waits until the last access of $\mathcal{A}$ to an edge $(u, v) \in B$ for which edge (x_{2k-2}, u) is already marked. Until this point, all queries to edges in B have negative outcomes. Only this last edge will have a positive outcome to an edge query, if it's made by $\mathcal{A}$, and the adversary will further make sure that this edge gets relaxed, before ending the phase.

To formalize this, let (u, v) be the edge accessed by $\mathcal{A}$ in the current operation. We describe the adversary's response by distinguishing several cases:

(s1) $(u, v) = (x_{2k-2}, v) \in A$. If this is a relaxation, mark (u, v). If this is an edge query do this: if (u, v) is unmarked, respond "yes", else respond "no".

(s2) $(u, v) \in B$. We have two sub-cases depending on the type of access.

Relaxation: If (x_{2k-2}, u) is marked, mark (u, v). If all edges in $A \cup B$ are marked, end phase k.

Edge query: If (x_{2k-2}, u) is not marked, respond "no". So suppose that (x_{2k-2}, u) is marked. In that case, if (u, v) is not the last unmarked edge in $A \cup B$, mark it and respond "no". If (u, v) is the last unmarked edge, respond "yes" (without marking).

(s3) $(u, v) \notin A \cup B$. If this is an edge query, respond "no". If this is a relaxation for (u, v), do nothing.

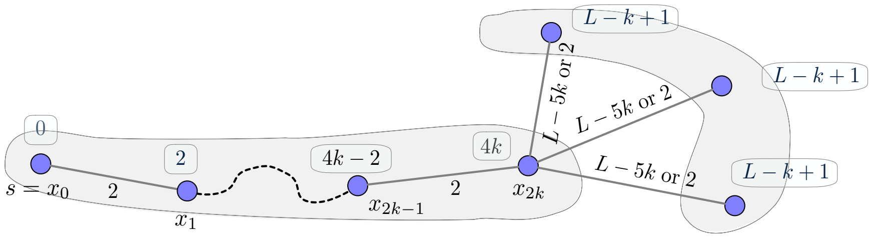


Figure 1 The state of the game right after phase k ends. Framed numbers next to vertices represent their D-values.

Let (u^*, v^*) be the edge marked last in this phase. This is the edge $(u, v) \in B$ from rule (s2) that becomes marked when it gets relaxed with all other edges in B already marked, ending the phase. At this point the adversary lets $x_{2k-1} = u^*$ and $x_{2k} = v^*$, and (if $k < (n-1)/2$) starts phase $k+1$.

For any permutation π of V starting with s, through the rest of the proof we denote by D_π the variable D-values produced by $\mathcal{A}$ when processing weight assignment ℓ_π. For each $k = 0, 1, ..., (n-1)/2$, the $(2k+1)$-permutation $x_0, x_1, ..., x_{2k}$ chosen by the adversary following the strategy above will be called the *kth cruel prefix*.

Invariant (I). We claim that the following invariant holds for each $k = 0, 1, ..., (n-1)/2$. Let $x_0, x_1, ..., x_{2k}$ be the adversary's kth cruel prefix. Then for each permutation π starting with $x_0, x_1, ..., x_{2k}$, the following properties hold when phase k of the adversary strategy ends (see Figure 1 for illustration):

(I0) All adversary's answers to $\mathcal{A}$'s queries in phases $1, 2, ..., k$ are correct for weight assignment ℓ_π.

(I1) $D_\pi[x_j] = 2j$ for $j = 0, 1, ..., 2k$.

(I2) If $w \in V \setminus \{x_0, x_1, ..., x_{2k}\}$ then $D_\pi[w] = L - k + 1$.

We postpone the proof of this invariant, and show first that it implies the $\Omega(n^3)$ lower bound. Indeed, from Invariant (I2) we conclude that the D-values will not represent the correct distances until after the last step of phase $(n-1)/2$. Since in each phase k all edges in the set $A \cup B$ for phase k will end up marked, the number of edge accesses in this phase is at least $|A \cup B| = |A| + |A|(|A| - 1) = |A|^2 = (n - 2k + 1)^2$. Thus, adding up the numbers of edge accesses in all phases $k = 1, 2, ..., (n-1)/2$, we obtain that the total number of steps in algorithm $\mathcal{A}$ is at least $(n-1)^2 + (n-3)^2 + ... + 2^2 = \frac{1}{6}n(n^2-1) = \Omega(n^3)$, giving us the desired lower bound.

It remains to prove Invariant (I). The invariant is true for $k = 0$, by the way the D-values are initialized. To argue that the invariant is preserved after each phase $k \geq 1$, we show that within this phase a more general invariant (J) holds, below.

Invariant (J). Let π be a permutation with prefix $x_0, x_1, ..., x_{2k}$, let $X_{k-1} = x_0, x_1, ..., x_{2k-2}$, and $Y_{k-1} = V \setminus X_{k-1}$. We claim that the following properties are satisfied during the phase, including right before and right after the phase.

(J0) All adversary's answers to $\mathcal{A}$'s queries up to the current step are correct for ℓ_π.

(J1) $D_\pi[x_j] = 2j$ for $j = 0, 1, ..., 2k - 2$.

(J2.1) If $w \in Y_{k-1} \setminus \{u^*, v^*\}$ then $$D_\pi[w] = \begin{cases} L - k + 2 & \text{if } (x_{2k-2}, w) \text{ unmarked} \\ L - k + 1 & \text{if } (x_{2k-2}, w) \text{ marked} \end{cases}$$

(J2.2) If $w = u^*$ then $$D_\pi[u^*] = \begin{cases} L-k+2 & \text{if } (x_{2k-2}, u^*) \text{ unmarked} \\ 4k-2 & \text{if } (x_{2k-2}, u^*) \text{ marked} \end{cases}$$

(J2.3) If $w = v^*$ then $$D_\pi[v^*] = \begin{cases} L-k+2 & \text{if } (x_{2k-2}, v^*) \text{ unmarked} \\ L-k+1 & \text{if } (x_{2k-2}, v^*) \text{ marked and } (u^*, v^*) \text{ unmarked} \\ 4k & \text{if } (u^*, v^*) \text{ marked} \end{cases}$$

When phase k starts, these properties are identical to Invariant (I) applied to the ending of phase $k-1$. We now show that these invariants are preserved within a phase k. Assume that the invariants hold up to some step, and consider the next operation when $\mathcal{A}$ accesses an edge (u, v). If $w \neq v$ then $D_\pi[w]$ is not affected, so assume that $w = v$. In the case analysis below, if the current step is a relaxation, we will use notation $D_\pi[v]$ for the D-value at v before this step and $D'_\pi[v]$ for the D-value at v after the step.

Case (s1). $(u, v) = (x_{2k-2}, v) \in A$. We consider separately the cases when this step is a relaxation or an edge query.

Relaxation: Suppose first that $v \neq u^*$. Then we have $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, v) = (4k-4) + (L - 5k + 5) = L - k + 1$. Thus, using (J2.1) and (J2.3), if (x_{2k-2}, v) was already marked then nothing changes, and if (x_{2k-2}, v) wasn't marked then $D'_\pi[v] = L - k + 1$, preserving (J2) because (x_{2k-2}, v) gets marked. (Note that in the special case $v = v^*$, edge (u^*, v^*) is not marked yet.)

For $v = u^*$ the argument is similar, except that now we use (J2.2): We have $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, u^*) = (4k-4) + 2 = 4k - 2$, so either (x_{2k-2}, u^*) is already marked and nothing changes, or $D'_\pi[u^*] = 4k - 2$ and (x_{2k-2}, u^*) gets marked.

Edge query: The reasoning here is analogous to the case of relaxation above. If $v \neq u^*$, then $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, v) = L - k + 1$ and the correctness of the adversary's answers follows from (J2.1) and (J2.3), If $v = u^*$, then $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, u^*) = (4k-4) + 2 = 4k - 2$, and the correctness of the adversary's answers follows from (J2.2).

Case (s2). $(u, v) \in B$. We consider separately the cases when this step is a relaxation or an edge query.

Relaxation: Suppose first that $u \neq u^*$. Then $D_\pi[u] \geq L - k + 1$, by (J2.1) and (J2.3). (This is true for the special case $u = v^*$, because (u^*, v^*) is not yet marked.) Since also $\ell(u, v) \geq 2$, this relaxation will not change the value of $D_\pi[v]$.

Next, consider the case $u = u^*$. If (x_{2k-2}, u^*) is unmarked then (J2.2) also implies (as in the previous sub-case) that the value of $D_\pi[v]$ will not change. So assume now that (x_{2k-2}, u^*) is marked, in which case $D_\pi[u^*] = 4k - 2$. If $v \neq v^*$ then the relaxation will not change the value of $D_\pi[v]$ because $\ell(u^*, v) = L$. For $v = v^*$, we have $D'_\pi[v^*] = D_\pi[u^*] + \ell(u^*, v^*) = (4k-2) + 2 = 4k$, preserving (J2.3), because this relaxation will mark (u^*, v^*).

Edge query: If (x_{2k-2}, u) is not marked then, by (J2.1)-(J2.3) we have $D_\pi[u] = L - k + 2$, and $\ell(u, v) \geq 2$, so the "no" answer by the adversary is correct.

Next, assume that (x_{2k-2}, u) is marked and $u \neq u^*$. Then $D_\pi[u] = L - k + 1$, by conditions (J2.1) and (J2.3) (since (u^*, v^*) is still not marked). So in this case the answer "no" is also correct.

The final case is when $u = u^*$ and (x_{2k-2}, u^*) is marked, so $D_\pi[u^*] = 4k - 2$. Now, if $v \neq v^*$ then the adversary responds "no", and since $\ell(u^*, v) = L$, this is correct. For $v = v^*$ we have $D_\pi[u^*] + \ell(u^*, v^*) = (4k-2) + 2 = 4k < D_\pi[v^*]$, where the last inequality is true because (u^*, v^*) is not marked. So the "yes" answer is also correct.

Case (s3). $(u, v) \notin A \cup B$. In this case we claim that $D_\pi[u] + \ell(u, v) \geq D_\pi[v]$, which implies the correctness for both cases, when this operation is a relaxation and edge update. The argument involves a few cases.

The first case is when $u \in Y_{k-1}$ and $v \in X_{k-1}$. Then we have $D_\pi[u] + \ell_{uv} \geq (4k-2) + 2 > D_\pi[v]$, applying (J1)-(J2.3).

If $u \in X_{k-1} \setminus \{x_{2k-2}\}$ and $v \in Y_{k-1}$ then there are two sub-cases, and in both we apply (J1) and (J2.1)-(J2.3). If $u = x_{2k-3}$ in which case $\ell(x_{2k-3}, v) = L$, the claim is trivial. If $u = x_j$ for $j \leq 2k-4$, then $D_\pi[x_j] = 2j$ and $\ell(x_j, v) \geq L - 5j/2$, so $D_\pi[u] + \ell(u, v) \geq (2j) + (L - 5j/2) = L - j/2 \geq L - k + 2 \geq D_\pi[v]$.

The final case is when $u, v \in X_{k-1}$, say $u = x_i$ and $v = x_j$. Here we use condition (J1). If $i > j$ then $D_\pi[x_i] > D_\pi[x_j]$. If $i < j-1$ then $\ell(x_i, x_j) \geq 2n$. If $i = j-1$ then $D_\pi[x_{j-1}] = 2j-2$, $D_\pi[x_j] = 2j$ and $\ell(x_{j-1}, x_j) = 2$. In each of these sub-cases, the claim holds.

The case analysis above completes the proof of invariants (J0)-(J2.3). By applying these invariants to the end of the phase, when all edges in $A \cup B$ are marked, gives us that invariant (I) holds after the phase, as needed – providing that the phase ends at all.

To complete the analysis of the adversary strategy we need to argue that phase k must actually end, in order for the D-values to represent correct distances from s. This follows directly from invariants (J1)-(J2.3), because they imply that before the very last step of the phase there is at least one vertex in Y_{n-1} with D-value at least $L - k + 1$, which is larger than its distance from s.

It now only remains to remove the assumption that the D-values are initialized to $L + 1$. According to our model, they need to be initialized to edge lengths from s, which are: $\ell(s, x_1) = 2$ and $\ell(s, v) = L$ for $v \neq x_1$ (since $s = x_0$). With this initialization, we only need to modify the first phase by marking all edges of the form (x_0, v) immediately. Invariant (J) then applies without further modification.

4 Lower Bound for Deterministic Algorithms with Three Types of Queries

In this section, we prove Theorem 2(a), an $\Omega(n^3)$ lower bound for deterministic algorithms using all three types of queries. Our argument is essentially a reduction – we show that any algorithm $\mathcal{A}$ that uses D-queries, weight queries, edge queries and relaxation updates can be converted into an algorithm $\mathcal{B}$ that has the same time complexity as $\mathcal{A}$ and uses only edge queries and relaxations. Our lower bound will then follow from Theorem 1(a).

We start with some initial observations that, although not needed for the proof, contain some useful insights. Since edge weights do not change, an algorithm can use weight queries to pre-sort all edges in time $O(n^2 \log n)$, and then it doesn't need to make any more weight queries during the computation. This way, the algorithm's running time is not affected as long as it's at least $\Omega(n^2 \log n)$. Similarly, the algorithm can use D-queries to maintain the total order of the D-values using, say, a binary search tree, paying a small overhead of $O(\log n)$ for each update operation. Then the algorithm's decisions at each step can as well depend on the total ordering of the vertices according to their current D-values. These changes will add at most an $O(\log n)$ factor to the running time.

Potential-oblivious model. Instead of working just with edge queries, we generalize our argument to *potential-oblivious* query models. We say that a query model $\mathbb{Q}$ is potential-oblivious if it satisfies the following property for each weight assignment ℓ and potential ϕ:

for any sequence of relaxations and queries from $\mathbb{Q}$ (with the D-values initialized as described in Section 2), the outcomes of the queries for weight assignments ℓ and $\ell + \Delta\phi$ are the same. By routine induction, any algorithm using a potential-oblivious model will perform the same sequence of queries and relaxations on assignments ℓ and $\ell + \Delta\phi$. Also, it will compute the correct distances on ℓ if and only if it will compute them for $\ell + \Delta\phi$, and in the same number of steps. (To see this, note that for each vertex v the invariant $D'[v] = D[v] + \phi(v)$ is preserved, where we use notations D and D' to distinguish between the D-values in the computations for ℓ and ℓ'. The respective distances $\ell(s,v)$ and $\ell'(s,v)$ satisfy the same equation.)

For example, the edge query only model is potential-oblivious. Due to our initialization and properties of relaxations, each value $D[v]$ always corresponds to the length of some path from s to v. Then the query is equivalent to comparing the length of two paths with the same start and end points, and the query outcome is the same after adding $\Delta\phi$, by the distance preservation property. Using these facts, potential-obliviousness follows from induction on the number of operations performed.

Golomb-ruler potential. For our proof, we need a potential function ϕ for which in the induced weight assignment $\Delta\phi$ all edge weights are different. (This naturally implies that all values of ϕ are also different.) Such an assignment is equivalent to a *Golomb ruler* (also known as a Sidon set), which is a set of non-negative integers with unique pair-wise differences. A simple Golomb ruler can be constructed using fast growing sequences, such as $\left\{2^i - 1\right\}_{i=0}^{n-1}$, but we are interested in sets contained in a small polynomial-in-n range. The asymptotic growth of Golomb rulers is well studied; it is known that there are n-element Golomb rulers that are subsets of $\{1, 2, ..., N\}$, for $N = n^2(1 + o(1))$ [10, 21], and that this bound on N is essentially optimal. Since the Golomb-ruler property is invariant under shifts, we can assume that a Golomb ruler contains number 0. For our purposes, this means that there exists a potential function ϕ that induces distinct edge weights with absolute maximum weight $O(n^2)$. ([4] shows that it is possible to obtain smaller maximum weights for certain classes of non-complete graphs, but this is not relevant to our constructions.) We will call this function a *Golomb-ruler potential.*

▶ **Theorem 3.** *Let $\mathcal{A}$ be a query/relaxation-based algorithm that uses relaxation updates, D-queries, weight queries and any queries from a potential-oblivious model $\mathbb{Q}$, and let $T(n)$ be the running time of $\mathcal{A}$. Then there is an algorithm $\mathcal{B}$ with running time $O(T(n))$ that uses only relaxation updates and queries from $\mathbb{Q}$.*

The idea of the proof is to convert a given weight assignment ℓ into another assignment ℓ' such that, if only queries from $\mathbb{Q}$ (and relaxations) are used, then (i) ℓ' is indistinguishable from ℓ using the queries from $\mathbb{Q}$, and (ii) in ℓ' the ordering of weights and the ordering of all D-values are *independent of ℓ* and, further, the ordering of the D-values is *fixed throughout the computation*, even though the D-values themselves may vary. $\mathcal{B}$ can do this conversion "internally" and simulate $\mathcal{A}$ on ℓ', and then it doesn't need to make any D-queries and weight queries, because their outcomes are predetermined.

Proof. Let $\mathcal{A}$ be a query/relaxation algorithm for that uses D-queries, weight queries, queries from $\mathbb{Q}$, and relaxation updates. We construct $\mathcal{B}$ that uses only queries from $\mathbb{Q}$ and relaxation updates. Let ϕ be the Golomb-ruler potential defined before the theorem. When run on a weight assignment ℓ, $\mathcal{B}$ will internally simulate $\mathcal{A}$ on weight assignment $\ell' = \ell + c\Delta\phi$, for $c = 2\ell_{\max}n + 1$. We use notation D' for the D-values computed by $\mathcal{A}$. The actions of $\mathcal{B}$ depend on the execution of $\mathcal{A}$ on ℓ', as follows:

- When $\mathcal{A}$ executes a weight query "$\ell'_{uv} < \ell'_{xy}$?", $\mathcal{B}$ directly executes the "yes" branch from the query if $\Delta\phi(u,v) < \Delta\phi(x,y)$, or the "no" branch otherwise.
- When $\mathcal{A}$ executes a D-query "$D'[u] < D'[v]$?", then $\mathcal{B}$ executes the "yes" branch if $\phi_u < \phi_v$, else it executes the "no" branch.

This simulation can be more formally described as converting the decision tree of $\mathcal{A}$ into the decision tree of $\mathcal{B}$. The tree of $\mathcal{B}$ is obtained by splicing out each node q representing a D-query or weight query. This splicing consists of connecting the parent of q to either the "yes" or "no" child of q, determined by the appropriate inequality involving ϕ, as explained above.

It remains to prove the correctness of $\mathcal{B}$. We argue first that $\mathcal{B}$ will produce correct distances if run on ℓ' instead of ℓ. For this, we observe that ℓ' satisfies the following properties:

(p1) For any two edges e, f, we have $\ell'_e < \ell'_f$ if and only if $\Delta\phi(e) < \Delta\phi(f)$.

(p2) For any three vertices u, x, y, any u-to-x path P_x and any u-to-y path P_y, we have $\ell'(P_x) < \ell'(P_y)$ if and only if $\phi_x < \phi_y$.

Indeed, both properties follow from the choice of c and straighforward calculation. For (p1), $\ell'_e < \ell'_f$ if and only if $\ell_e - \ell_f < c[\Delta\phi(f) - \Delta\phi(e)]$, and because $|\ell_e - \ell_f| < c$ this inequality is determined by the sign of $\Delta\phi(f) - \Delta\phi(e)$, which is always non-zero, by the Golomb-ruler property. (Note that here we only use that $c > 2\ell_{\max}$.) The justification for (p2) is similar: we have $\ell'(P_x) = \ell(P_x) + c(\phi_x - \phi_u)$ and $\ell'(P_y) = \ell(P_y) + c(\phi_y - \phi_u)$, so $\ell'(P_x) < \ell'(P_y)$ if and only if $\ell(P_x) - \ell(P_y) < c[\phi_y - \phi_x]$, and since $|\ell(P_x) - \ell(P_y| < c$ this inequality is determined by the sign of $\phi_y - \phi_x$.

Properties (p1) and (p2) imply that when we run $\mathcal{A}$ on ℓ', in each weight query we can equivalently use assignment $\Delta\phi$ instead of ℓ', and instead of using a D-query we can compare the corresponding potential values. Therefore $\mathcal{B}$ works correctly for ℓ'. But since now $\mathcal{B}$ uses only relaxations and queries from $\mathbb{Q}$, that are potential-oblivious, and ℓ' is obtained from ℓ by adding a weight assignment induced by potential $c\phi$, $\mathcal{B}$'s computation on ℓ will also be correct. ◀

Theorem 3, together with Theorem 1 implies the $\Omega(n^3)$ lower bound for query/update-based algorithms that use D-queries, weight queries, any set of potential-oblivious queries, and relaxation updates. Since the edge update is potential oblivious, Theorem 2(a) follows.

Further, using the construction from Theorem 2(a), where a weight assignment with maximum weight $O(n)$ was used, the proof of Theorem 3 shows that Theorem 2(a) holds even if all weights are bounded by $O(n^4)$.

A side result for general graphs. The reduction in the proof of Theorem 3 extends naturally to arbitrary graphs. In particular, we can extend a result from Eppstein [9]:

▶ **Theorem 4.** *For any n and m where $n \le m \le n(n-1)$, there exists a graph with n nodes and m edges where any deterministic algorithm $\mathcal{A}$ using D-queries, weight queries, and relaxation updates has worst-case running time $\Omega(nm/\log n)$. If $m = \Omega(n^{1+\varepsilon})$ for some $\varepsilon > 0$, the lower bound can be improved to $\Omega(nm)$.*

Proof sketch. We focus on the case where m is arbitrary. First note that the model using no queries and relaxation updates is equivalent to non-adaptive algorithms (that is, universal relaxation sequences) described in [9]. (It's also obvious that the query model using no queries is potential-oblivious.) Let G be the graph construction described in the proof of [9, Theorem 3]. In particular, G has n nodes and m edges, and for every non-adaptive algorithm on G, there is weight assignment that forces $\Omega(nm/\log n)$ relaxations. If there exists an algorithm $\mathcal{A}$ for G using D-queries, weight queries, and relaxation updates that runs in $T(n)$ time, then by the same construction as in Theorem 3, there also exists an algorithm $\mathcal{B}$ using only relaxation updates that runs in time $O(T(n))$. Then an $o(nm/\log n)$ running time on G would contradict the lower-bound on non-adaptive algorithms. The proof for the case $m = \Omega(n^{1+\varepsilon})$ is identical. ◀

As explained in Section 5, the reduction also applies to randomized algorithms, and because [9] proves the same lower bounds for expected running time for randomized non-adaptive algorithms, the above bounds also apply to the randomized case.

5 Lower Bounds for Randomized Algorithms

In this section, we extend the proofs in Sections 3 and 4 to obtain $\Omega(n^3)$ lower bounds for randomized algorithms, proving Theorem 1(b) and Theorem 2(b). The proofs are based on Yao's principle [22]: we give a probability distribution on weight assignments for which the expectation of each deterministic algorithm's running time is $\Omega(n^3)$.

We fix the value of n. Let $\ell_{\max}$ be the maximum absolute value of weights used in the proof, whose value will be specified later. Let $\mathbb{L}$ be the family of all weight assignments $\ell : E \to [-\ell_{\max}, \ell_{\max}]$. Denote by $\mathbb{A}$ the (finite) set of all deterministic query/relaxation-based algorithms with running time at most $2n^3$. We only need to consider algorithms in $\mathbb{A}$, because any other algorithm in our model can be modified to run in time at most $2n^3$. To see why, consider this algorithm's decision tree. For any node at depth n^3, replace its subtree by the Bellman-Ford relaxation sequence. The resulting tree remains correct, and its depth is at most $2n^3$.

For a deterministic algorithm $\mathcal{A} \in \mathbb{A}$ and weight assignment $\ell \in \mathbb{L}$, denote by $T(\mathcal{A}, \ell)$ the running time of $\mathcal{A}$ on assignment ℓ. Let $\Pi(\mathbb{A})$ be the set of all probability distributions on $\mathbb{A}$ and $\Pi(\mathbb{L})$ be the set of all probability distributions on $\mathbb{L}$. Any randomized algorithm $\mathcal{R}$ is simply a probability distribution on $\mathbb{A}$, so $\mathcal{R} \in \Pi(\mathbb{A})$. Denote by $\mathrm{Exp}_{x\sim\theta} f(x)$ the expected value of $f(x)$, for a random variable x from distribution θ. The lemma below is a restatement of Yao's principle [22] in our context:

▶ **Lemma 5.** *The following equality holds:*

$$\min_{\mathcal{R}\in\Pi(\mathbb{A})} \max_{\ell\in\mathbb{L}} \mathrm{Exp}_{\mathcal{A}\sim\mathcal{R}} T(\mathcal{A}, \ell) \;=\; \max_{\sigma\in\Pi(\mathbb{L})} \min_{\mathcal{A}\in\mathbb{A}} \mathrm{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell).$$

In this lemma, both sides involve the expected running time, with the difference being that on the left-hand side we consider randomized algorithms and their worst-case inputs, while the right-hand side involves the probability distribution on *input permutations* that is worst for *deterministic algorit hms.*

Proof of Theorem 1(b) (Sketch).[1] We give a probability distribution σ on weight assignments ℓ for which every deterministic algorithm needs $\Omega(n^3)$ steps in expectation to compute correct distances. This is sufficient, as then Lemma 5 implies that each randomized algorithm $\mathcal{R}$ makes $\Omega(n^3)$ steps in expectation on some weight assignment.

[1] A detailed proof will appear in the full version of this paper.

Recall that in the proof of Theorem 1(a) in Section 3 we used weight assignments associated with permutations of vertices. This is also the case here, although this assignment needs to be modified. For any permutation $\pi = x_0, x_1, ..., x_{n-1}$ of the vertices, the corresponding weight assignment is

$$\ell_\pi(x_i, x_j) \;=\; \begin{cases} n & \text{if } j = i+1 \\ L - (n+\frac{1}{2})i & \text{if } j \geq i+2 \text{ and } i \text{ is even} \\ L & \text{if } j \geq i+2 \text{ and } i \text{ is odd} \end{cases}$$

where L is sufficiently large, say $5n^2$. (We explain later why larger weights are necessary.) In our argument here, the adversary chooses the uniform distribution σ on all $(n-1)!$ permutations π starting with s.

In a certain sense, our goal now is simpler than in Section 3, as the adversary's job, which is to choose σ, is already done. We "only" need to lower bound the expected running time of algorithms from $\mathbb{A}$ if the weights are distributed according to σ. The challenge is that this argument needs to work for *an arbitrary algorithm* from $\mathbb{A}$.

So fix any deterministic algorithm $\mathcal{A} \in \mathbb{A}$. We need to prove that $\mathrm{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell) = \Omega(n^3)$. A high-level approach in our proof is similar to the proof in Section 3: we partition the computation of $\mathcal{A}$ into $(n-1)/2$ phases, and show that the expected length of each phase $k = 1, 2, ..., (n-1)/2$ is $\Omega((n-2k)^2)$.

For a specific permutation $\pi = x_0, x_1, ..., x_{n-1}$ with $x_0 = s$ and $k = 1, 2, ..., (n-1)/2$, let $t_k(\pi)$ be the first time step such that in steps $1, 2, ..., t_k(\pi)$ the edges $(x_0, x_1), ..., (x_{2k-1}, x_{2k})$ have been accessed by $\mathcal{A}$ (that is, relaxed or queried) in this particular order. We refer to the time interval $(t_{k-1}(\pi), t_k(\pi)]$ as *phase k for permutation π*.

The proof idea is this: In each phase k of the strategy in Section 3 the adversary was able to force the algorithm to relax all edges from x_{2k-2} to Y before revealing edge (x_{2k-1}, x_{2k}), thus ensuring that at all times all D-values differ at most by 1. This is not possible anymore, because now the algorithm can get "lucky" and relax edges (x_{2k-2}, x_{2k-1}) and (x_{2k-1}, x_{2k}) before all edges (x_{2k-2}, u) for $u \in Y$ are relaxed, and then the D-values for such vertices u will reflect the relaxations that occurred in some earlier phases. But we can still bound the differences between D-values. Namely, by our choice of the length function above, any two D-values will differ by at most $n/2$. Thus, since all edge lengths are at least n, the negative answers to all edge queries inside Y are still correct, independently of the suffix $x_{2k-1}, ..., x_{n-1}$ of π.

More specifically, the analysis is based on establishing two invariants, captured by the claim below (formal proof omitted here).

▷ Claim 6. The following invariants are satisfied when each phase k starts:

(R1) The computation of $\mathcal{A}$ up until phase k starts is independent of the suffix $x_{2k-1}, x_{2k}, ..., x_{n-1}$ of π.

(R2) The D-values have the following form: $D[x_j] = jn$ for $j \leq 2k-2$, and $D[x_j] \in [L-k+1, L]$ for $j \geq 2k-1$.

Next, define $\tilde{t}_k$ to be a random variable whose values are $t_k(\pi)$ for permutations π distributed randomly according to σ. We refer to the time interval $(\tilde{t}_{k-1}, \tilde{t}_k]$ as *phase k*, and let $\partial t_k = \tilde{t}_k - \tilde{t}_{k-1}$ be the random variable equal to the length of this phase.

We then prove the following claim:

▷ Claim 7. $\mathrm{Exp}_{\ell\sim\sigma}[\partial t_k] \geq \frac{1}{2}(n-2k+1)(n-2k+2)$.

Let $\bar{z} = z_0, z_1, ...z_{2k-2}$ be some fixed $(2k-1)$-permutation of V with $z_0 = s$. Let H be the event that π starts with $\bar{z}$. It is sufficient to prove the inequality in Claim 7 for the conditional expectation $\text{Exp}_{\ell\sim\sigma}[\partial t_k | H]$.

So assume that event H is true. Let $Y = V \setminus \{z_0, ..., z_{2k-2}\}$. Now the argument is this: The suffix $x_{2k-1}, ..., x_{n-1}$ of π is a random permutation of Y and the edge (x_{2k-1}, x_{2k}) is uniformly distributed among the edges in Y. Algorithm $\mathcal{A}$ is deterministic and all edge queries for edges inside Y, except for edge (x_{2k-1}, x_{2k}) (and only if (x_{2k-2}, x_{2k-1}) has already been relaxed), will have negative answers. Similarly, all queries to edges from x_{2k-2} to Y will have positive answers. So $\mathcal{A}$ will be accessing these edges in some order that is uniquely determined by the state of $\mathcal{A}$ when phase k starts. Since there are $(n-2k+1)(n-2k+2)$ edges in Y, this implies that on average it will take $\frac{1}{2}(n-2k+1)(n-2k+2)$ steps for $\mathcal{A}$ to access (x_{2k-1}, x_{2k}), even if we don't take into account that (x_{2k-2}, x_{2k-1}) needs to be accessed first. This will imply Claim 7.

We now continue the proof of Theorem 1(b). For the algorithm to be correct, if the chosen permutation π is $x_0, x_1, ..., x_{n-1}$, then the algorithm needs to relax the edges on this path in order as they appear on the path. So its running time is at least $t_{(n-1)/2}(\pi)$. Since $\tilde{t}_{(n-1)/2} = \sum_{k=1}^{(n-1)/2} \partial t_k$, using Claim 7 and applying the linearity of expectation we obtain that $\text{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell) \geq \text{Exp}_{\ell\sim\sigma}[\tilde{t}_{(n-1)/2}] = \sum_{k=1}^{(n-1)/2} \text{Exp}_{\ell\sim\sigma}[\partial t_k] = \Omega(n^3)$, completing the proof. ◀

Proof of Theorem 2(b). There is not much to prove here, because the reduction described in Section 4 applies with virtually no changes to randomized algorithms. Indeed, just like in the proof of Theorem 2(a) (or more specifically the proof of Theorem 3), suppose that $\mathcal{R}$ is a randomized algorithm that uses all three types of queries: D-queries, weight-queries, the queries from model $\mathbb{Q}$, as well as relaxation updates, and let $T(n)$ be $\mathcal{R}$'s expected running time. We can convert $\mathcal{R}$ into a randomized algorithm $\mathcal{R}'$ with running time $O(T(n))$ that uses only the queries from $\mathbb{Q}$ and relaxation updates. With this, Theorem 2(b) follows from Theorem 1(b). ◀

High-probability bounds. Using standard reasoning (see [9], for example), our lower bound results for expectation imply respective high-probability bounds, namely that there are no randomized algorithms in the models from Theorems 1 and 2 that compute correct distance values in time $o(n^3)$ with probability at least $1 - o(1)$.

To justify this, suppose that $\mathcal{R}$ is a randomized algorithm that computes correct distance values in time $T(n) = o(n^3)$ with probability $1 - o(1)$. Consider the algorithm $\mathcal{R}'$ obtained from $\mathcal{R}$ by switching to the Bellman-Ford relaxation sequence right after step $T(n)$. The expected running time of $\mathcal{R}'$ is then at most $T(n) + o(1)n^3 = o(n^3)$, but this would contradict our lower bounds in Theorems 1(b) and 2(b).

6 Final Comments and Open Problems

Our reduction in Section 4 introduces negative weights, raising a natural question: *Is it possible to use $o(n^3)$ relaxations with only weight-queries for instances with non-negative weights?* (This question is of purely theoretical interest, because $O(n^2)$ relaxations can be achieved, using Dijkstra's algorithm, if D-queries are used instead.) Our proof techniques do not work for this variant. The reason is, in the instances we construct the shortest-path tree is a Hamiltonian path, and for such instances this path can be uniquely determined by the weight ordering: start from s, and at each step follow the shortest outgoing edge from the current vertex to a yet non-visited vertex. So only $n-1$ relaxations are needed. It is unclear what is the "hard" weight ordering in this case. It can be shown that in the two extreme cases: (i) if the weight orderings of outgoing edges from each vertex are agreeable (that is, they are determined by a permutation of the vertices), or (ii) if they are random, then there is a relaxation sequence of length only $O(n^{2.5})$ (and this likely can be improved further).

A natural extension of our query/relaxation model would be to allow *unconditional edge updates* of the form $D[v] \leftarrow D[u] + \ell_{uv}$. A combination of such edge updates and D-queries allows an algorithm to check for properties that are impossible to test if only relaxation updates are used. For example, by applying edge updates repeatedly around cycles, such an algorithm would be able to determine, for any given rational number c, whether one cycle is at least c times longer than some other cycle.

A more open-ended question is to determine if there are simple types of queries, say some linear inequalities involving weights and the D-values (with a constant number of variables), that would be sufficient to yield an adaptive algorithm (possibly randomized) that makes $o(n^3)$ relaxations.

The case of random universal sequences is also not fully resolved. While it is known that the asymptotic bound is $\Theta(n^3)$, there is a factor-of-4 gap for the leading constant, between $\frac{1}{12}$ and $\frac{1}{3}$ [9, 1].

We remark that our proofs are somewhat sensitive to the initialization of the D-values. Recall that in our model we assume that initially $D[v] = \ell_{sv}$ for $v \neq s$. This is natural, and it guarantees that at all times the D-values represent lengths of paths from s. It also has the property of being language- and platform-independent. However, some descriptions of shortest-path algorithms initialize the D-values to infinity, or some very large number. The proof of Theorem 1 in Section 3 can be modified to work if the D-values were initialized to some sufficiently large value M (the adversary can then use $L = M - 1$ in her strategy). However, then the edge lengths are no longer polynomial, and the proof of Theorem 2 in Section 4 does not apply in its current form. Initializing to infinity would also affect the proofs. The reduction in Section 4 can be modified to account for infinite D-values, but we don't know how to adapt the proof in Section 3 to this model. We leave open the problem of finding a more "robust" lower bound proof, that works for an arbitrary valid initialization and uses only polynomial weights.

References

1 Michael J. Bannister and David Eppstein. Randomized speedup of the Bellman-Ford algorithm. In *Proceedings of the 9th Meeting on Analytic Algorithmics and Combinatorics, ANALCO 2012*, pages 41–47. SIAM, 2012. doi:10.1137/1.9781611973020.6.

2 Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

3 Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 600–611, 2022. doi:10.1109/FOCS54457.2022.00063.

4 Béla Bollobás and Oleg Pikhurko. Integer sets with prescribed pairwise differences being distinct. *European Journal of Combinatorics*, 26(5):607–616, 2005. doi:10.1016/J.EJC.2004.04.008.

5 Gang Cheng and Nirwan Ansari. Finding all hops shortest paths. *IEEE Commun. Lett.*, 8(2):122–124, 2004. doi:10.1109/LCOMM.2004.823365.

6 Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log W)$ time (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 752–771, 2017. doi:10.1137/1.9781611974782.48.

7 Narsingh Deo and Chi-Yin Pang. Shortest-path algorithms: Taxonomy and annotation. *Networks*, 14(2):275–323, 1984. doi:10.1002/NET.3230140208.

8 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.

9 David Eppstein. Lower bounds for non-adaptive shortest path relaxation. In *Proceedings of the 18th International Symposium on Algorithms and Data Structures, WADS 2023*, pages 416–429, 2023. doi:10.1007/978-3-031-38906-1_27.

10 P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941.

11 Jeremy T. Fineman. Single-source shortest paths with negative real weights in $\tilde{O}(mn^{8/9})$ time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 3–14, 2024. doi:10.1145/3618260.3649614.

12 L. R. Ford. *Network Flow Theory.* RAND Corporation, Santa Monica, CA, 1956.

13 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

14 Roch Guérin and Ariel Orda. Computing shortest paths for any number of hops. *IEEE/ACM Trans. Netw.*, 10(5):613–620, 2002. doi:10.1109/TNET.2002.803917.

15 Jialu Hu and László Kozma. Non-adaptive Bellman-Ford: Yen's improvement is optimal. *CoRR*, abs/2402.10343, 2024. arXiv:2402.10343, doi:10.48550/arXiv.2402.10343.

16 Stasys Jukna and Georg Schnitger. On the optimality of Bellman-Ford-Moore shortest path algorithm. *Theor. Comput. Sci.*, 628:101–109, 2016. doi:10.1016/J.TCS.2016.03.014.

17 Tomasz Kociumaka and Adam Polak. Bellman-Ford is optimal for shortest hop-bounded paths. In *Proceedings of the 31st Annual European Symposium on Algorithms, ESA 2023*, pages 72:1–72:10, 2023. doi:10.4230/LIPICS.ESA.2023.72.

18 Ulrich Meyer, Andrei Negoescu, and Volker Weichert. New bounds for old algorithms: On the average-case behavior of classic single-source shortest-paths approaches. In *Proceedings of the First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems, TAPAS 2011*, pages 217–228, 2011. doi:10.1007/978-3-642-19754-3_22.

19 E. F. Moore. The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching, Part II*, pages 285–292, 1959.

20 A. Shimbel. Structure in communication nets. In *Proceedings of the Symposium on Information Networks*, pages 199–203. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955.

21 James Singer. A theorem in finite projective geometry and some applications to number theory. *Trans. Amer. Math. Soc.*, 43(3):377–385, 1938.

22 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

23 Y. Yen. *Shortest Path Network Problems*, volume 18 of *Mathematical Systems in Economics.* Verlag Anton Hain, Meisenheim am Glan, 1975.

Fault-Tolerant Bounded Flow Preservers

Shivam Bansal
Department of Computer Science and Engineering, IIT Delhi, India

Keerti Choudhary
Department of Computer Science and Engineering, IIT Delhi, India

Harkirat Dhanoa
Department of Computer Science and Engineering, IIT Delhi, India

Harsh Wardhan
Department of Electrical Engineering, IIT Delhi, India

Abstract

Given a directed graph $\mathcal{G} = (V, E)$ with n vertices, m edges and a designated source vertex $s \in V$, we consider the question of finding a sparse subgraph $\mathcal{H}$ of $\mathcal{G}$ that preserves the flow from s up to a given threshold λ even after failure of k edges. We refer to such subgraphs as (λ, k)-fault-tolerant bounded-flow-preserver ((λ, k)-FT-BFP). Formally, for any $F \subseteq E$ of at most k edges and any $v \in V$, the (s, v)-max-flow in $\mathcal{H} \setminus F$ is equal to (s, v)-max-flow in $\mathcal{G} \setminus F$, if the latter is bounded by λ, and at least λ otherwise. Our contributions are summarized as follows:

1. We provide a polynomial time algorithm that given any graph $\mathcal{G}$ constructs a (λ, k)-FT-BFP of $\mathcal{G}$ with at most $\lambda 2^k n$ edges.
2. We also prove a matching lower bound of $\Omega(\lambda 2^k n)$ on the size of (λ, k)-FT-BFP. In particular, we show that for every $\lambda, k, n \geqslant 1$, there exists an n-vertex directed graph whose optimal (λ, k)-FT-BFP contains $\Omega(\min\{2^k \lambda n, n^2\})$ edges.
3. Furthermore, we show that the problem of computing approximate (λ, k)-FT-BFP is NP-hard for any approximation ratio that is better than $O(\log(\lambda^{-1} n))$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Fault-tolerant Data-structures, Max-flow, Bounded Flow Preservers

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.9

Related Version *Full Version*: https://arxiv.org/abs/2404.16217

Funding *Keerti Choudhary*: The author is supported in part by Google India Algorithms Research grant 2021.

1 Introduction

We address the problem of computing single-source fault-tolerant bounded-flow-preservers for directed graphs. The objective is to construct a sparse subgraph that preserves the flow value up to a parameter λ from a given fixed source s, even after failure of up to k edges.

The following definition provides a precise characterization of this subgraph.

▶ **Definition 1.** *Let $\mathcal{G} = (V, E)$ be a directed graph with unit edge-capacities and $s \in V$ be a designated source vertex. A (λ, k)-Fault-Tolerant Bounded-Flow-Preserver ((λ, k)-FT-BFP) for $\mathcal{G}$ is a subgraph $\mathcal{H} = (V, E_{\mathcal{H}} \subseteq E)$ of $\mathcal{G}$ satisfying that for every $F \subseteq E$ of at most k edges, and every $t \in V$,*

$$\textsc{max-flow}(s,t,\mathcal{H}-F) = \begin{cases} \textsc{max-flow}(s,t,\mathcal{G}-F) & \text{if } \textsc{max-flow}(s,t,\mathcal{G}-F) \leqslant \lambda, \\ \text{At least } \lambda, & \text{otherwise.} \end{cases}$$

For the special case of $\lambda = 1$, the problem is referred to as k-Fault-Tolerant Reachability Subgraph (k-FTRS) in the literature. Here the goal is to preserve reachability from s after k edge failures. Baswana et al. [4] showed that there exists a k-FTRS with at most $2^k n$ edges. Lokshtanov et al. [17] presented an algorithm for computing a (λ, k)-FT-BFP for directed graphs. Their algorithm runs in time $O(4^{k+\lambda}(k+\lambda)^2(m+n) \cdot m)$, and each vertex of the FT-BFP has in-degree at most $4^{k+\lambda}(k+\lambda)$. They also showed that a $(k+\lambda-1)$-FTRS of a graph $\mathcal{G}$ also serves as it's (λ, k)-FT-BFP. Using this result in conjunction with the algorithm from [4], they obtain an alternate construction of a (k, λ)-FT-BFP with at most $2^{k+\lambda}n$ edges. However, this bound is quadratic in n for any λ larger than $\log n$.

We consider the problem of obtaining a tight bound on (λ, k)-FT-BFP. Specifically, we aim to answer the following question:

Given a directed graph $\mathcal{G} = (V, E)$ with a source s, and a flow threshold $\lambda \geqslant \log n$, can we construct a sparse *(λ, k)-FT-BFP $\mathcal{H} = (V, E_H \subset E)$? If so, can we present graphs for which the construction turns out to be tight?*

In this paper, we affirmatively answer the question above. We provide construction for FT-BFP that has a linear dependence on λ.

1.1 Upper Bound Results and Applications

We prove the following:

▶ **Theorem 2.** *There exists an algorithm that for any directed graph $\mathcal{G}$ on n vertices and m edges, and any integers $\lambda, k \geqslant 1$, computes in $O(\lambda 2^k mn)$ time a (λ, k)-FT-BFP for $\mathcal{G}$ with at most $\lambda 2^k n$ edges.*

We also present an application of our FT-BFP construction in computing an all-pairs fault-tolerant λ-reachability oracle. We show that for any positive constants $\lambda, k \geqslant 1$, we can compute an oracle of $O(n^2)$ size that given any query vertex-pair $x, y \in V$ and any set F of k edge failures, reports (x, y)-λ-reachability in $\mathcal{G} \setminus F$ efficiently.

▶ **Theorem 3.** *Given any directed graph $\mathcal{G} = (V, E)$ on n vertices and any positive constants $\lambda, k \geqslant 1$, we can preprocess $\mathcal{G}$ in polynomial time to build an $O(n^2)$ size data structure that, given any query vertex-pair (x, y) and any set F of k edges, reports the (x, y) λ-reachability in $\mathcal{G} \setminus F$ in $O(n^{1+o(1)})$ time.*

1.2 Lower Bound and Hardness Results

We show that the extremal bound of $\lambda 2^k n$ obtained in Theorem 2 is tight. In particular, we prove existence of n-vertex graphs whose (λ, k)-FT-BFP must contain at least $\Omega(\min \lambda 2^k n, n^2)$ edges.

▶ **Theorem 4.** *For every $\lambda, k, n \geqslant 1$ satisfying $\lambda 2^k = O(n)$, there exists a construction of an n-vertex directed graph whose optimal (λ, k)-FT-BFP contains $\Omega(\lambda 2^k n)$ edges.*

While the lower-bound in above theorem proves that the bound of $\lambda 2^k n$ obtained in Theorem 2 is existentially tight, it does not address the problem of computing a sparsest (λ, k)-FT-BFP.

We next demonstrate the hardness of computing optimal (λ, k)-FT-BFP structures. We show that unless $P = NP$, there is no polynomial-time algorithm to obtain an $O(\log(\lambda^{-1}n))$-approximation to optimal (λ, k)-FT-BFP.

▶ **Theorem 5.** *For any $\lambda, k, n \geqslant 1$ satisfying $k = \Omega(\log(\lambda^{-1}n))$, the problem of computing an $O(\log(\lambda^{-1}n))$ approximation to optimal (λ, k)-FT-BFP for n vertex directed graphs is NP-hard.*

As a corollary, we obtain the following hardness result for the FTRS problem.

▶ **Corollary 6.** *For any $k, n \geqslant 1$ satisfying $k = \Omega(\log n)$, the problem of computing an $O(\log n)$ approximation to optimal k-FTRS for n vertex directed graphs is NP-hard.*

1.3 Existing Works

For undirected graphs, there exists a tight construction for (λ, k)-FT-BFP with $O((k+\lambda)\cdot n)$ edges that directly follows from edge connectivity certificate constructions provided by Nagamochi and Ibaraki [19].

A closely related problem to that of graph preservers is fault-tolerant reachability oracles. For dual failures, the work of [11] obtained an $O(n)$ size single source reachability oracle with constant query time for directed graphs. Brand and Saranurak [23], showed construction of an $\widetilde{O}(n^2)$ sized k-fault-tolerant all-pairs reachability oracle that has $O(k^\omega)$ query time, where ω is the constant of matrix multiplication.

Recently, Baswana et al. [2] considered the problem of constructing a sensitivity oracle for reporting the max-flow value for a single source-destination pair. They presented an $O(n^2)$ size data-structure that after failure of any two edges, reports the max-flow value of the surviving graph in constant time.

For the problem of computing the value of all-pairs max-flow up to λ in the static setting, Abboud et at. [1] obtained two deterministic algorithms that work for DAGs: a combinatorial algorithm which runs in $O(2^{O(\lambda^2)}\cdot mn)$ time, and another algorithm that can be faster on dense graphs which runs in $O((\lambda \log n)4^{\lambda+o(\lambda)}\cdot n^\omega)$ time.

Some other graph theoretic problems studied in the fault-tolerant model include computing distance preservers [12, 21, 20], depth-first-search tree [3], spanners [8, 13], approximate single source distance preservers [5, 22, 6], approximate distance oracles [14, 9], compact routing schemes [9, 7].

2 Preliminaries

Given a digraph $\mathcal{G} = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges with unit edge capacities, we first define some notations used throughout the paper.

- IN$(v, \mathcal{G})$: The set of in-neighbours of v in $\mathcal{G}$.
- OUT$(v, \mathcal{G})$: The set of out-neighbours of v in $\mathcal{G}$.
- IN-EDGES$(v, \mathcal{G})$: The set of all incoming edges of v in $\mathcal{G}$.
- OUT-EDGES$(v, \mathcal{G})$: The set of all outgoing edges of v in $\mathcal{G}$.
- OUT$(A, \mathcal{G})$: The set of all those vertices in $V \backslash A$ having an incoming edge from some vertex of A in $\mathcal{G}$, where $A \subseteq V(\mathcal{G})$.
- $\mathcal{G}(A)$: The subgraph of $\mathcal{G}$ induced by the vertices lying in a subset A of V.
- $\mathcal{G} + (u, v)$: The graph obtained by adding an edge (u, v) to graph $\mathcal{G}$.
- $\mathcal{G} \setminus F$: The graph obtained by deleting the edges lying in a set F from graph $\mathcal{G}$.
- MAX-FLOW$(S, t, \mathcal{G})$: The value of the maximum flow in graph $\mathcal{G}$ from a source set S to a destination vertex t. When the set S comprises of a single vertex, say s, we represent it simply by MAX-FLOW$(s, t, \mathcal{G})$.
- PATH$[a, b, T]$: The path from node a to b in a tree T.

- $P[a,b]$: The subpath of path P lying between vertices a and b, where a precedes b on P.
- $P \circ Q$: The path formed by concatenating paths P and Q in $\mathcal{G}$. Here it is assumed that the last edge (or vertex) of P is the same as the first edge (or vertex) of Q.

We next define the concept of farthest min-cut that was introduced by Ford and Fulkerson in their pioneering work on flows and cuts [15]. Let S be a source set, and t be a destination vertex. Any (S,t)-cut C is a partition of the vertex set into two sets: $A(C)$ and $B(C)$, where $S \subseteq A(C)$ and $t \in B(C)$. An (S,t)-min-cut C^* is said to be the *farthest min-cut* if $A(C^*) \supsetneq A(C)$ for every (S,t)-min-cut C other than C^*. We denote the cut C^* by $\mathrm{FMC}(S,t,\mathcal{G})$. Similar to farthest-min-cut, we can define the nearest min-cut. An (S,t)-min-cut C^* is said to be the *nearest min-cut* if $A(C^*) \subsetneq A(C)$ for every (S,t)-min-cut C other than C^*. We denote the cut C^* by $\mathrm{NMC}(S,t,\mathcal{G})$.

Below we state a property of nearest and farthest (s,t)-min-cuts [15] showing that they can be computed efficiently.

▶ **Property 7.** *Let s be a source vertex, t be a destination vertex, and f be an s to t max-flow in graph $\mathcal{G}$. Let $\mathcal{G}_f$ denote the residual graph corresponding to flow f. Further let X be the set of vertices reachable from s in $\mathcal{G}_f$, and Y be the set of vertices having a path to t in $\mathcal{G}_f$. Then $NMC(s,t,\mathcal{G}) = (X, V \setminus X)$ and $FMC(s,t,\mathcal{G}) = (V \setminus Y, Y)$.*

3 Hardness of logarithmic approximation

We prove in this section the following hardness result for approximating optimal FT-BFP.

▶ **Theorem 8.** *For any $\lambda, k, n \geqslant 1$ satisfying $k = \Omega(\log(\lambda^{-1}n))$, the problem of computing an $O(\log(\lambda^{-1}n))$ approximate (λ,k)-FT-BFP for n vertex digraphs is NP-hard.*

We prove the above theorem by showing a reduction from the SET-COVER problem to the optimal FT-BFP.

▶ **Problem 9** ([18], Definition 1)**.** *The input to SET-COVER consists of base set U, $|U| = n$ and a family $\mathfrak{F} = (S_1, ..., S_m)$ of m subsets of U satisfying $\cup_{j=1}^{m} S_j = U$, $m \leqslant poly(n)$. The goal is to find as few sets $S_{i_1}, ..., S_{i_k}$ as possible that cover U, that is, $\cup_{j=1}^{k} S_{i_j} = U$*

▶ **Lemma 10** ([18], Theorem 2)**.** *For every $0 < \alpha < 1$ (exact) SAT on inputs of size n can be reduced in polynomial time to approximating SET-COVER to within $(1-\alpha)\ln N$ on inputs of size $N = n^{O(1/\alpha)}$.*

From Lemma 10, we can also deduce that it is NP-Complete to approximate SET-COVER up to a multiplicative factor of $c_1 \log \max(n,m)$ for some $c_1 > 0$ as $m \leqslant poly(n)$.

Transformation. Given a SET-COVER instance $\langle U, \mathfrak{F} \rangle$, we will construct a (λ,k)-FT-BFP instance $\langle \mathcal{G}, s \rangle$. The transformation is as follows (also see Figure 1).

1. Round up the number for elements in U to nearest power of 2 (let this be 2^u) by adding $2^u - |U|$ new elements to U and all these new elements to every set in $\mathfrak{F}$.
2. Initialize $\mathcal{G}$ to be the graph with $N+1$ vertices, namely, $s, v_1, \ldots, v_N$ where $N = 4\lambda(m+n)$.
3. Next construct the following subgraph $\mathcal{G}_i$, for each $i \in [1,\lambda]$.
   a. Construct a complete binary tree B_i rooted at a vertex r_i of height u and 2^u leaf nodes. The leaf nodes of B_i will correspond to elements in the universe U. From each leaf node x_i in B_i, add out-edges to two new vertices, namely, $\ell(x_i)$ and $r(x_i)$.

   b. For each set $W \in \mathfrak{F}$, add a vertex $y_{i,W}$ to graph $\mathcal{G}_i$. Let Y_i denote the resulting set which consists of $|\mathfrak{F}|$ vertices. For each $x \in U$ and $W \in \mathfrak{F}$, add an edge from $\ell(x_i)$ to $y_{i,W}$ if and only if $x \in W$.

   c. Add a set Z_i of $u+1$ additional vertices. For each leaf x_i in B_i, add an edge from $r(x_i)$ to each vertex in the set Z_i.

4. Finally, we add an edge from s to the roots $r_1, \dots, r_\lambda$. Also for each $i \in [1, \lambda]$, we add an edge from each vertex in $Y_i \cup Z_i$ to each of the vertices $v_1, \dots, v_N$.

We set $k = u + 1$ for this (λ, k)-FT-BFP instance.

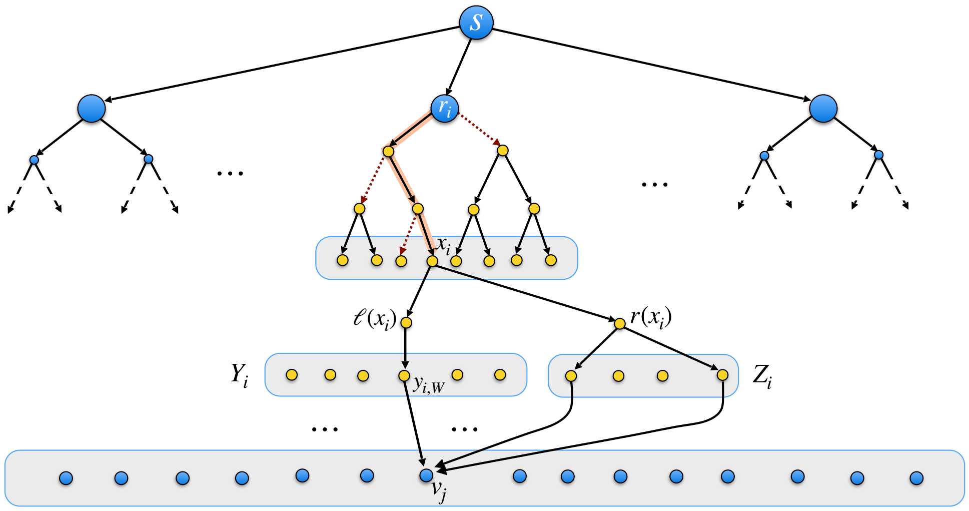


Figure 1 Depiction of a (λ, k)-FT-BFP instance obtained from a SET-COVER instance $\langle U, \mathfrak{F} \rangle$.

▶ **Lemma 11.** *Any (λ, k)-FT-BFP $\mathcal{H}$ of the graph instance $\langle \mathcal{G}, s \rangle$, can be used to construct a solution of the SET-COVER instance of size at most $\lambda^{-1}(\min_{j=1}^{N} |\text{IN}(v_j, \mathcal{H})|)$.*

Proof. Consider a vertex v_j in $\mathcal{H}$ that minimizes $|\text{IN}(v_j, \mathcal{H})|$. Consider the following candidate solutions

$$S_i = \{W \in \mathfrak{F} \mid (y_{i,W}, v_j) \in E(\mathcal{H})\}.$$

Out of the λ sets, namely $S_1, \dots, S_\lambda$, let S_{i_0} be the set with least cardinality. The cardinality of S_{i_0} is at most $|\text{IN}(v_j, \mathcal{H})|/\lambda$ as minimum value is upper-bounded by the average value.

Now in order to prove that S_{i_0} is a valid solution, consider an element $x \in U$. Let P be the unique path from r_{i_0} to leaf node x_{i_0} in B_{i_0}, and let F_1 be the set of all those edges $(u, v) \in B_{i_0}$ such that $u \in P$ and v is the child of u not lying on P. Observe that x_{i_0} is the unique leaf in B_{i_0} that is reachable from s in $\mathcal{H} \setminus F_1$. Let F_2 be a singleton set comprising of the edge $(x_{i_0}, r(x_{i_0}))$. Consider the set $F = F_1 \cup F_2$ of size k. Since $\text{MAX-FLOW}(s, v_j, \mathcal{G} \setminus F) = \lambda$, there must exists a path, say Q, from s to v_j in $\mathcal{H} \setminus F$ passing through r_{i_0}. Such a path Q must pass through $\ell(x_{i_0})$ as well as a vertex in Y_{i_0}, say $y_{i_0,W}$. This implies that the edge $(y_{i_0,W}, v_j)$ lies in $\mathcal{H}$, and so by definition of S_{i_0}, the set W lies in S_{i_0}. Moreover W contains the element x as $(\ell(x_{i_0}), y_{i_0,W})$ is an edge in $\mathcal{G}$. This proves that element $x \in U$ is covered by S_{i_0}, and thus S_{i_0} is a valid solution to $\langle U, \mathfrak{F} \rangle$. ◀

▶ **Lemma 12.** *Any solution S of the SET-COVER instance $\langle U, \mathfrak{F} \rangle$, can be used to construct a solution $\mathcal{H}$ of (λ, k)-FT-BFP instance satisfying $|\text{IN}(v_j, \mathcal{H})| = \lambda(|S| + k)$, for each $j \in [1, N]$.*

Proof. Let S be a solution of the SET-COVER instance $\langle U, \mathfrak{F}\rangle$. Consider the sets

$$A_i = \{y_{i,W} \mid W \in S\} \cup Z_i, \text{ for } i \leqslant \lambda, \quad \text{and} \quad A = \bigcup_{i=1}^{\lambda} A_i.$$

We will show that

$$\mathcal{H} = \mathcal{G} \setminus \cup_{j=1}^{N} \textsc{In-Edges}(v_j) + \cup_{j=1}^{N} (A \times v_j).$$

is a (λ, k)-FT-BFP of $\mathcal{G}$.

Let us assume, to the contrary, that $\mathcal{H}$ is not a (λ, k)-FT-BFP of $\mathcal{G}$. Then there must exist an edge set F of size at most k and an index $j \in [1, N]$ satisfying $\textsc{max-flow}(s, v_j, \mathcal{G} \setminus F)$ is greater than $\textsc{max-flow}(s, v_j, \mathcal{H} \setminus F)$. Observe that each path from s to v_j must pass through a vertex r_i, for some $i \in [1, \lambda]$, and each r_i only allows a unit flow to pass through it.

Since $\textsc{max-flow}(s, v_j, \mathcal{G} \setminus F) > \textsc{max-flow}(s, v_j, \mathcal{H} \setminus F)$, there must exist an index $i \in [1, \lambda]$ satisfying that there exists a path from s to v_j in $\mathcal{G} \setminus F$ *passing through* r_i, but no such corresponding path exists in $\mathcal{H} \setminus F$.

Let $R = \{x_i^0, x_i^1, \ldots, x_i^\alpha\}$ be the set of leaf nodes in tree B_i *reachable* from s in $\mathcal{G} \setminus F$. There exist at least $\min(k+1, |R|)$ vertex-disjoint paths from R to v_j in $\mathcal{H}$, namely,

- $(\{x_i^0, \ell(x_i^0), y_{i,W}, v_j)$, where $W \in \mathfrak{F}$ is the set in S that contains the element $x^0 \in U$.
- $(\{x_i^c, r(x_i^c), z_i^c, v_j)$, for $c = 1$ to $\min(k, |R| - 1)$.

Thus even after k faults atleast one path from r_i to v_j will exist in $\mathcal{H} \setminus F$. This contradicts the assumption that there is no s to v_j path in $\mathcal{G} \setminus F$ passing through r_i. Hence, $\textsc{max-flow}(s, v_j, \mathcal{G} \setminus F)$ must be identical to $\textsc{max-flow}(s, v_j, \mathcal{H} \setminus F)$. ◀

The proof of Theorem 8 now directly follows from Lemma 10, Lemma 11, and Lemma 12, along with the fact that for every integer $n \geqslant 1$, there exist hard instances of the SET-COVER problem $(U, \mathfrak{F})$ satisfying $|U| = n$, where the size of the optimal solution is significantly larger than $\log |U|$.

4 Upper bound of $\lambda 2^k n$ Edges

In this section we will provide construction of a sparse (λ, k)-FT-BFP.

4.1 Locality Property for Flow Preservers

▶ **Lemma 13.** *Let $\mathcal{G} = (V, E)$ be a graph with a source $s \in V$, $\lambda \geqslant 1$ be an integer, and v be a fixed vertex in V. Let $\alpha = \min\big(\lambda, \textsc{max-flow}(s, v, \mathcal{G})\big)$. Let $\mathcal{E}_v$ be the set of in-edges of v corresponding to any arbitrary set of α-edge-disjoint paths from s to v in $\mathcal{G}$. Further, let $\mathcal{H}$ be a subgraph of $\mathcal{G}$ obtained by restricting the in-edges of the given node v to those present in $\mathcal{E}_v$. Then, for each $t \in V$, we have*

$$\textsc{max-flow}(s, t, \mathcal{H}) \;\geqslant\; \min\big(\lambda, \textsc{max-flow}(s, t, \mathcal{G})\big).$$

Proof. We first observe that $\alpha = \textsc{max-flow}(s, v, \mathcal{H})$. Indeed, by construction there are at least α edge-disjoint paths from s to v in $\mathcal{H}$, additionally, the in-degree of v in $\mathcal{H}$ is exactly α which proves that the (s, v)-max-flow in $\mathcal{H}$ can not be larger than α.

Now consider a vertex $t \in V$, and let $\beta = \textsc{max-flow}(s, t, \mathcal{H})$. Consider an (s, t)-min-cut (A, B) in $\mathcal{H}$. If $v \in A$ then, by construction of $\mathcal{H}$, the (s, t)-cut (A, B) has value β also in $\mathcal{G}$, so $\beta \geqslant \textsc{max-flow}(s, t, \mathcal{G})$ and we are done. Assume next $v \in B$. Then (A, B) is an (s, v)-cut of value β in $\mathcal{H}$. Since $\alpha = \textsc{max-flow}(s, v, \mathcal{H})$, we have $\beta \geqslant \alpha$. If $\alpha = \lambda$ we are done. We next study the **non-trivial case** of $\alpha = \textsc{max-flow}(s, v, \mathcal{G}) < \lambda$.

Let f be an (s,t)-max-flow in $\mathcal{H}$. Let us assume on contrary that $\beta < \textsc{max-flow}(s,t,\mathcal{G})$. Then the residual graph $\mathcal{G}_f$ must have an augmenting path, say P, containing some edges present in $\mathcal{G}$ but not in $\mathcal{H}$. Such edges must be all incoming to v. Thus, $P = P[s,w] \circ (w,v) \circ P[w,t]$ where $(w,v) \in E(G) \setminus E(H)$, and $P[s,w], P[v,t]$ are present in the residual graph $\mathcal{H}_f$. Adding P to f gives an (s,t)-flow of in $\mathcal{H} + (w,v)$, implying that

(i) $\textsc{max-flow}(s,t,\mathcal{H} + (w,v)) = \beta + 1$

(ii) $(w,v) \in A \times B$

(iii) (A,B) is an (s,t)-min-cut in $\mathcal{H} + (w,v)$

Let $\{Q_i \circ e_i \circ Q'_i\}_{i=1}^{\alpha}$ be α edge-disjoint s-to-v paths in $\mathcal{H}$, where the edge e_i of each such path is its last edge crossing the (s,v)-cut (A,B), so $V(Q'_i) \subseteq B$. Such exist as $\alpha = \textsc{max-flow}(s,v,\mathcal{H})$. Let $e_{\alpha+1}, \ldots, e_\beta$ be the other edges crossing (A,B) in $\mathcal{H}$. Let $e_0 = (w,v)$, crossing (A,B) by (ii). Let $\{P_j \circ e_j \circ P'_j\}_{j=0}^{\beta}$ be $\beta+1$ edge-disjoint s-to-t paths in $\mathcal{H} + (w,v)$, each crossing the cut (A,B) exactly once, at e_j, so $V(P_j) \subseteq A$. Such exist by (i) and (iii). Then, $\{P_0 \circ e_0\} \cup \{P_i \circ e_i \circ Q'_i\}_{i=1}^{\alpha}$ are $\alpha+1$ edge-disjoint s-to-v paths in $\mathcal{G}$, contradicting $\alpha = \textsc{max-flow}(s,v,\mathcal{G})$. ◀

In the next lemma we show that in order to compute a sparse (λ,k)-FT-BFP it suffices to focus on a single destination node.

▶ **Lemma 14** (Locality Lemma for Flow Preservers). *Let $\mathcal{A}$ be an algorithm that given any graph $\mathcal{G}$ and any vertex $v \in V(\mathcal{G})$, computes a (λ,k)-FT-BFP of $\mathcal{G}$ with at most $c_{\lambda,k}$ in-edges to v. Then using $\mathcal{A}$, one can construct for any n vertex digraph a (λ,k)-FT-BFP with at most $c_{\lambda,k} \cdot n$ edges.*

Proof. Consider a graph $\mathcal{G}$ with n vertices, namely, $v_1, \ldots, v_n$. We will provide a construction of (λ,k)-FT-BFP of $\mathcal{G}$ using black-box access to algorithm $\mathcal{A}$. We compute a sequence of graphs $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_n$ as follows:

1. Initialize $\mathcal{G}_0 = \mathcal{G}$.
2. For $i \geqslant 1$, compute $\mathcal{G}_i$ in two steps:
   a. First use $\mathcal{A}$ to compute a (λ,k)-FT-BFP of $\mathcal{G}_{i-1}$ in which the in-degree of v_i is bounded by $c_{\lambda,k}$, let this graph be $\mathcal{H}_{i-1}$.
   b. Obtain $\mathcal{G}_i$ from $\mathcal{G}_{i-1}$ by restricting the incoming edges of v_i to those present in $\mathcal{H}_{i-1}$.

It is easy to verify that the in-degree of each vertex in $\mathcal{G}_n$ is at most $c_{\lambda,k}$.

To show that $\mathcal{G}_n$ is a (λ,k)-FT-BFP of $\mathcal{G}$, it suffices to show that $\mathcal{G}_i$ is a (λ,k)-FT-BFP of $\mathcal{G}_{i-1}$, for each $i \geqslant 1$.

Let us fix an index i in the range $[1,n]$. Consider a set F of at most k edges in $\mathcal{G}_{i-1}$, and let

$$\alpha = \min\big(\lambda, \textsc{max-flow}(s, v_i, \mathcal{G}_{i-1} \setminus F)\big).$$

By construction, $\mathcal{H}_{i-1}$ is a (λ,k)-FT-BFP of $\mathcal{G}_{i-1}$, so there exists at least α edge-disjoint paths from s to v_i in the graph $\mathcal{H}_{i-1} \setminus F$. Let $\mathcal{E}_i$ be the set of in-edges of v_i corresponding to these α edge-disjoint paths. Observe that the edges in $\mathcal{E}_i$ lie in graph $\mathcal{G}_i \setminus F$. Moreover, graphs $\mathcal{G}_i \setminus F$ and $\mathcal{G}_{i-1} \setminus F$ differ only at in-edges of v_i. Therefore, by Lemma 13 it follows that for any vertex $t \in V(G)$, $\textsc{max-flow}(s,t,\mathcal{G}_i \setminus F) \geqslant \min\big(\lambda, \textsc{max-flow}(s,t,\mathcal{G}_{i-1} \setminus F)\big)$. This proves that $\mathcal{G}_i$ is a (λ,k)-FT-BFP of $\mathcal{G}_{i-1}$. ◀

4.2 Construction of an Improved FTRS

We present here an improved bound on the in-degree of a node t in an FTRS when the node t satisfies that (s,t)-max-flow in $\mathcal{G}$ is larger than one. In particular, we prove the following theorem.

► **Theorem 15.** *Let $\mathcal{G}$ be an n vertex, m edges directed graph with a designated source node s. Let t be a vertex satisfying $\textsc{max-flow}(s,t,\mathcal{G}) = f$, for some positive integer f. Then for every $k \geqslant 1$, we can compute in $O(2^k fm)$ time a $(k+f-1)$-FTRS for $\mathcal{G}$ in which the in-degree of node t is at most $2^k f$.*

Let us focus on a single destination node t. We first show that it suffices to provide construction of $(k+f-1)$-FTRS for a graph in which out-degree of each vertex other than s is bounded by 2. In order to prove this we will transform the graph $\mathcal{G} = (V,E)$ into another graph $\mathcal{H} = (V_H, E_H)$ satisfying that (i) the value of (s,t)-max-flow in graphs $\mathcal{G}$ and $\mathcal{H}$ is identical; (ii) the out-degree of every vertex in $\mathcal{H}$ other than s is bounded by two. The steps to transform $\mathcal{G}$ into graph $\mathcal{H}$ are as follows:

1. Initialize $\mathcal{H}$ to be the graph $\mathcal{G}$.
2. Split each edge $e = (x,y) \in E$ by inserting two new vertices $\ell_{x,y}$ and $r_{x,y}$ between the endpoints x and y, so that edge (x,y) is translated into the path $(x, \ell_{x,y}, r_{x,y}, y)$.
3. For every node $y \in V \setminus \{s,t\}$ if $x_1,\ldots,x_p$ are in-neighbours of y in $\mathcal{G}$ and $z_1,\ldots,z_q$ are out-neighbours of y in $\mathcal{G}$, then we replace vertex y (in current $\mathcal{H}$) by p binary trees as follows. First we remove node y from $\mathcal{H}$. Next for each $x_i \in \textsc{in}(y,\mathcal{G})$ insert a binary tree $B_{x_i,y}$ to graph $\mathcal{H}$ (along with new internal nodes and edges) whose root is $r_{x_i,y}$ and leaves are $\ell_{y,z_1},\ldots,\ell_{y,z_q}$.

Notice that $\mathcal{H}$ has $O(mn)$ edges and vertices. Indeed for every vertex v (other than s and t) in $\mathcal{G}$, $|\textsc{in}(v,\mathcal{G})|$ binary trees have been added to $\mathcal{H}$, each of size $O(|\textsc{out}(v,\mathcal{G})|)$. So the number of edges and vertices in the transformed graph is $O(\sum_{v\in V} |\textsc{in}(v,\mathcal{G})| \cdot |\textsc{out}(v,\mathcal{G})|)) = O(mn)$. Also, observe that the out-degree of each vertex in $\mathcal{H}$ other than s bounded by two.

► **Lemma 16.** *$\textsc{max-flow}(s,t,\mathcal{G}) = \textsc{max-flow}(s,t,\mathcal{H})$*

Proof. We will show that each s to t path in $\mathcal{G}$ now corresponds to a unique s to t path in $\mathcal{H}$. Suppose there exists a path $(s = u_0, u_1, u_2, \ldots, u_k = t)$ in $\mathcal{G}$. Then we will have an equivalent path in $\mathcal{H}$ as

$$\begin{aligned}(s, \ell_{u_0,u_1}, r_{u_0,u_1}) \circ \textsc{path}(r_{u_0,u_1}, \ell_{u_1,u_2}, B_{u_0,u_1}) \circ (\ell_{u_1,u_2}, r_{u_1,u_2}) \circ \ell \cdots \circ \\ \textsc{path}(r_{u_{k-2},u_{k-1}}, \ell_{u_{k-1},u_k}, B_{u_{k-1},u_k}) \circ \ell_{u_{k-1},u_k}, r_{u_{k-1},u_k}) \circ (r_{u_{k-1},u_k}, t)\end{aligned}$$

where $\textsc{path}(r,\ell,B)$ denotes the path from r to ℓ using edges in binary tree B. Therefore, the (s,t)-max-flow values in graphs $\mathcal{G}$ and $\mathcal{H}$ are identical. ◄

We will now justify the significance of our transformation by providing a way to construct a $(k+f-1)$-FTRS of $\mathcal{G}$ if we know a $(k+f-1)$-FTRS for $\mathcal{H}$ such that the in-degree of t in both the FTRSs is identical.

► **Lemma 17.** *A $(k+f-1)$-FTRS for $\mathcal{G}$ can be constructed by knowing a $(k+f-1)$-FTRS of $\mathcal{H}$, that preserves the in-degree of node t.*

Proof. Let $\mathcal{H}^*$ be a $(k+f-1)$-FTRS of $\mathcal{H}$. We want to construct $\mathcal{G}^*$, a $(k+f-1)$-FTRS for $\mathcal{G}$ satisfying the condition that in-degree of t in graphs $\mathcal{G}^*$ and $\mathcal{H}^*$ is identical.

The construction of $\mathcal{G}^*$ is as follows: For each in-neighbour w of the vertex t in $\mathcal{G}$, include edge (w, t) in $\mathcal{G}^*$ if and only if edge $(r_{w,t}, t)$ is present in $\mathcal{H}^*$. Thus, the in-degree of t in graphs $\mathcal{G}^*$ and $\mathcal{H}^*$ is identical. For vertices v other than t, we include all in-neighbours of v in $\mathcal{G}^*$.

We will now prove that $\mathcal{G}^*$ is a $(k + f - 1)$-FTRS of $\mathcal{G}$. Consider any set F of at most k failed edges in $\mathcal{G}$. Define a set F_0 of failed edges in $\mathcal{H}$ by including edge $(\ell_{u,v}, r_{u,v})$ in F_0 for every $(u, v) \in F$. From the path correspondence above and the fact that $\mathcal{H}^*$ is a $(k + f - 1)$-FTRS of $\mathcal{H}$, it is evident that for any $r \leqslant \lambda$, there are r-edge-disjoint paths from s to t in $\mathcal{G}^* \setminus F$ if and only if there are r-edge-disjoint paths from s to t in $\mathcal{H}^* \setminus F_0$. Therefore, $\mathcal{G}^*$ is a $(k + f - 1)$-FTRS of $\mathcal{G}$. ◀

It was shown in [4] that if out-degree of s is one, and out-degree of all other vertices is bounded by two, then Algorithm 1 computes a k-FTRS for $\mathcal{G}$ in which in-degree of t is at most 2^k. We will prove in the next lemma that if MAX-FLOW$(s, t, \mathcal{G}) = f$, and out-degree of every vertex other than s is bounded by two, then Algorithm 1 in fact computes a $(k + f - 1)$-FTRS for $\mathcal{G}$ in which the in-degree of t is at most $2^k f$.

▶ **Lemma 18.** *Let $\mathcal{G}$ be a directed graph satisfying that the out-degree of every vertex other than the designated source s is bounded by 2, and $k \geqslant 1$ be an integer parameter. Let $t \in V(\mathcal{G})$ satisfy* MAX-FLOW$(s, t, \mathcal{G}) = f$*, for some positive integer f. Then Algorithm 1 computes a $(k + f - 1)$-FTRS for $\mathcal{G}$ in which the in-degree of node t is at most $2^k f$.*

Proof. Consider the following algorithm from [4] for computing k-FTRS that bounds in-degree of an input node t.

Algorithm 1 Algorithm for computing k-FTRS.

```
$S_1 \leftarrow \{s\}$;
for $i = 1$ to $k$ do
    $C_i \leftarrow \mathrm{FMC}(S_i, t, \mathcal{G})$;
    $(A_i, B_i) \leftarrow \mathrm{Partition}(C_i)$;
    $S_{i+1} \leftarrow (A_i \cup \mathrm{OUT}(A_i, \mathcal{G})) \setminus \{t\}$;
end
$f_0 \leftarrow$ max-flow from $S_{k+1}$ to $t$;
$\mathcal{E}(t) \leftarrow$ Incoming edges of $t$ present in $E(f_0)$;
Return $\mathcal{G}^* = (\mathcal{G} \setminus \textsc{In-Edges}(t, \mathcal{G})) + \mathcal{E}(t)$;
```

We will now show $\mathcal{G}^*$ is a $(k + f - 1)$-FTRS of $\mathcal{G}$. Let F be any set of $k + f - 1$ failed edges. If there exists a path R from s to t in $\mathcal{G} \setminus F$ then we shall prove the existence of a path $\hat{R}$ from s to t in $\mathcal{G}^* \setminus F$. Observe that R must pass through each (s, t)-cut C_i, for each $i \in [1, k]$, through an edge, say (u_i, v_i). If $v_i = t$ then $(u_i, v_i) \in \mathcal{E}(t)$ and thus R is intact in the graph $\mathcal{G}^*$. Now we need to prove for the case when the edge $(u_i, v_i) \notin \mathcal{E}(t)$.

To prove that a path $\hat{R}$ exists in $\mathcal{G}^*$, we will construct a sequence of auxiliary graphs as done in [4], say $\mathcal{H}_i$'s, for each $i \in [1, k + 1]$, as follows:

$$\mathcal{H}_1 = \mathcal{G}, \qquad \mathcal{H}_i = \mathcal{G} + (s, v_1) + \ldots + (s, v_{i-1}), i \in [2, k + 1].$$

From the induction proof of Lemma 18 of [4], we get MAX-FLOW$(s, t, \mathcal{H}_{i+1}) = 1 +$ MAX-FLOW$(s, t, \mathcal{H}_i)$ and since MAX-FLOW$(s, t, \mathcal{H}_1) =$ MAX-FLOW$(s, t, \mathcal{G}) = f$, we get that MAX-FLOW$(s, t, \mathcal{H}_{k+1}) = k + f$. Let $\mathcal{H}^* = (\mathcal{H}_{k+1} \setminus \textsc{In-Edges}(t)) + \mathcal{E}(t)$ i.e. the incoming edges of t are restricted in $\mathcal{H}_{k+1}$ to those present in the set $\mathcal{E}(t)$. In Lemma 19 of [4] it is shown that $\text{MAX-FLOW}(s, t, \mathcal{H}^*) = \text{MAX-FLOW}(s, t, \mathcal{H}_{k+1}) = k + f$. Since the flow in $\mathcal{H}^*$ is greater than $|F|$ or the number of faults, we can directly use the Lemma 20 of [4] to see that there exists a path $\hat{R}$ in $\mathcal{G}^* \setminus F$.

The bound on the number of edges also follows from [4]. Lemma 21 of [4] states that $|C_{i+1}| \leqslant 2|C_i|$ where $C_{k+1} = FMC(S_{k+1}, t, \mathcal{G})$. Since $|C_1| = f$, we get the bound on $\mathcal{E}(t) = C_{k+1}$ as $2^k f$. Note that the proof of Lemma 21 of [4] assumes that every vertex has out-degree bounded by two but it can be shown that the Lemma will hold true even when the out-degree of all vertices except the source vertex is bounded by two by using the fact that in the proof of Lemma 21, $\text{OUT}(A_i)$ will never contain the source vertex for any i. ◀

4.3 Computing sparse (λ, k)-FT-BFP

In this subsection, we will show how to construct a (λ, k)-FT-BFP of $\mathcal{G}$ from a $(k + f - 1)$-FTRS of $\mathcal{G}$. We will start by introducing a lemma from [17], followed by additional lemmas that will help us to obtain a tight construction for FT-BFP.

▶ **Lemma 19** ([17]). *Let $\mathcal{G}$ be a directed graph with a designated source node s, and let $\mathcal{H}$ be a $(k + \lambda - 1)$-FTRS of $\mathcal{G}$. Then, $\mathcal{H}$ is also a (λ, k)-FT-BFP of $\mathcal{G}$.*

To strengthen the above lemma, we present a method for constructing a (λ, k)-FT-BFP from a $(\min\{f, \lambda\} + k - 1)$-FTRS, where f represents the maximum flow from the source node s to a destination node t in the graph.

▶ **Lemma 20.** *Let $\mathcal{G}$ be a directed graph with a designated source node s, and let t be a vertex satisfying $\text{MAX-FLOW}(s, t, \mathcal{G}) = f$, for some positive integer f. Then a $(\min\{f, \lambda\} + k - 1)$-FTRS of $\mathcal{G}$ that differs from $\mathcal{G}$ only at in-edges of t is a (λ, k)-FT-BFP for $\mathcal{G}$.*

Proof. Let $\mathcal{H}$ be a $(\min\{f, \lambda\} + k - 1)$-FTRS of $\mathcal{G}$ that deviates from $\mathcal{G}$ only at in-edges of t. It follows from Lemma 19 that the subgraph $\mathcal{H}$ is a $(\min\{f, \lambda\}, k)$-FT-BFP for $\mathcal{G}$.

The claim trivially holds true if $f \geqslant \lambda$, so let us consider the scenario $f < \lambda$. Consider a set F of at most k edge failures in $\mathcal{G}$, and let p be $\text{MAX-FLOW}(s, t, \mathcal{G} \setminus F)$. Since $p \leqslant f < \lambda$ and $\mathcal{H}$ is a (f, k)-FT-BFP, the max-flow from s to t in $\mathcal{H} \setminus F$ must be exactly p.

Since $\mathcal{G}$ and $\mathcal{H}$ only differs at in-edges of t, it follows from Lemma 13 that for each $v \in V(\mathcal{G})$, $\text{MAX-FLOW}(s, v, \mathcal{H} \setminus F) \geqslant \min(\lambda, \text{MAX-FLOW}(s, v, \mathcal{G} \setminus F))$. This proves that $\mathcal{H}$ is a (λ, k)-FT-BFP for $\mathcal{G}$. ◀

We now provide construction of a (λ, k)-FT-BFP that bounds the in-degree of a single destination node t.

▶ **Lemma 21.** *Let $\mathcal{G}$ be an n vertex, m edges directed graph with a designated source node s, and t be any arbitrary vertex in $\mathcal{G}$. Then for any $\lambda, k \geqslant 1$, we can compute in $O(\lambda 2^k m)$ time a (λ, k)-FT-BFP for $\mathcal{G}$ in which the in-degree of t is bounded above by $\lambda 2^k$.*

Proof. Let f be the value of (s, t)-max-flow in $\mathcal{G}$. We present a construction of a (λ, k)-FT-BFP, say $\mathcal{H}$, by considering the following two cases.

Case 1. max-flow$(s, t, \mathcal{G}) \geqslant \lambda + k$:
Let us start by taking a look at the scenario $f \geqslant \lambda + k$. In this case we can choose any $\lambda + k$ incoming edges of t which carry a flow of $\lambda + k$ from s to t and discard all other incoming edges of t to construct $\mathcal{H}$. The resulting graph $\mathcal{H}$ will be a (λ, k)-FT-BFP of $\mathcal{G}$ due to Lemma 20, and the in-degree of t in $\mathcal{H}$ will be $\lambda + k \leqslant \lambda 2^k$.

Case 2. max-flow$(s,t,\mathcal{G}) < \lambda + k$:
We next consider the case $f < \lambda + k$. In this case we use Theorem 15 to compute a $(\min\{f,\lambda\} + k - 1)$-FTRS of $\mathcal{G}$, say $\mathcal{H}_0$, such that the in-degree of t in $\mathcal{H}_0$ is at most $2^k \min\{f,\lambda\}$. We obtain the graph $\mathcal{H}$ from $\mathcal{G}$ by limiting the incoming edges of t to those present in $\mathcal{H}_0$. The resulting graph $\mathcal{H}$ will be a (λ,k)-FT-BFP of $\mathcal{G}$ due to Lemma 20. ◀

We conclude with the following theorem that directly follows by combining together Lemma 14 and Lemma 21.

▶ **Theorem 22.** *Let $\mathcal{G}$ be an n vertex, m edges directed graph with a designated source node s. Then for any $\lambda, k \geqslant 1$, we can compute in $O(\lambda 2^k mn)$ time a (λ,k)-FT-BFP for $\mathcal{G}$ with at most $\lambda 2^k n$ edges. Moreover, the in-degree of each vertex in this (λ,k)-FT-BFP is bounded above by $\lambda 2^k$.*

5 Matching Lower Bound

We shall now show that for each λ, k, n ($n \geqslant 3\lambda 2^{k+1}$), there exists a directed graph $\mathcal{G}$ with $O(n)$ vertices whose (λ,k)-FT-BFP must have $\Omega(2^k \lambda n)$ edges.

The construction of graph $\mathcal{G}$ is as follows. Let $B_1,\ldots,B_\lambda$ be vertex-disjoint complete binary trees of height k rooted at vertices $r_1,\ldots,r_k$, and let s be a new vertex have an edge to each of the r_i's. Let X denote the set of leaf nodes of these λ trees, and let Y be another set containing $n - (1 + \sum_{i=1}^{\lambda} |V(B_i)|)$ vertices. Note that $|Y| \geqslant n/3$. The graph $\mathcal{G}$ is obtained by adding an edge from each $x \in X$ to each $y \in Y$. In other words, $V(\mathcal{G}) = \{s\} \cup V(B_1) \cup \cdots \cup V(B_\lambda) \cup Y$ and $E(\mathcal{G}) = \{(s,r_i) \mid 1 \leqslant i \leqslant \lambda\} \cup E(B_1) \cup \cdots \cup (B_\lambda) \cup (X \times Y)$.

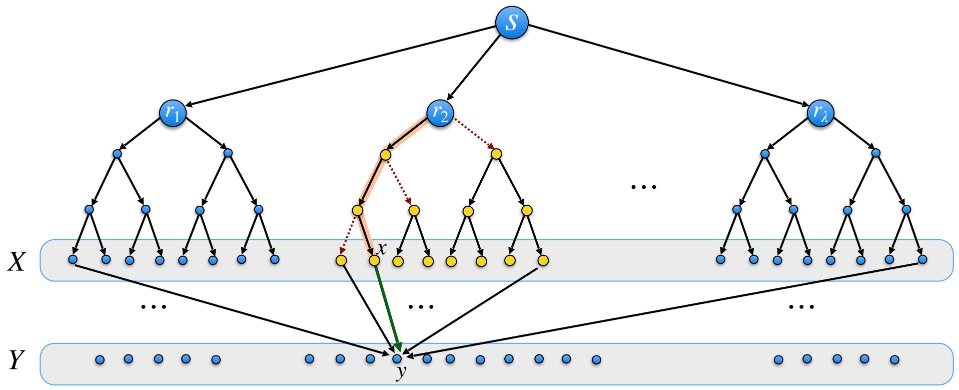


Figure 2 Depiction of lower bound on the size of (λ,k)-FT-BFP when $k = 3$.

We prove in the following lemma that any (λ,k)-FT-BFP of the above constructed graph contains at least $\Omega(2^k \lambda n)$ edges.

▶ **Lemma 23.** *Any (λ,k)-FT-BFP of $\mathcal{G}$ must contain $\Omega(2^k \lambda n)$ edges.*

Proof. It is easy to see that the out-edges of s, and the edges of each of the binary tree B_i's must be present in a (λ,k)-FT-BFP of $\mathcal{G}$. Thus, let us consider an edge $(x,y) \in X \times Y$, where x is the leaf node of some binary tree B_i.

Let P be the unique path from r_i to x in B_i, and let F be the set of all those edges $(u,v) \in B_i$ such that $u \in P$ and v is the child of u not lying on P. On failure of set F, there remains a unique path from s to y that passes through edge (s, r_i). Moreover, $\textsc{max-flow}(s, y, \mathcal{G} \setminus F) = \lambda$. So, any subgraph $\mathcal{H}$ of $\mathcal{G}$ not containing (x, y) edge would not be a (λ, k)-FT-BFP as on failure set F, $\mathcal{H}$ would not preserve (s, y)-max-flow.

Hence, any (λ, k)-FT-BFP of $\mathcal{G}$ contains at least $|X \times Y| = 2^k\lambda|Y| \geqslant 2^k\lambda n/3$ edges. ◀

6 Applications

In this section we present applications of FT-BFP structure.

6.1 Fault-tolerant All-Pairs λ-reachability oracle

Georgiadis et al. [16] showed that for any n vertex directed graph $\mathcal{G} = (V, E)$ we can compute 2-reachability information for all pairs of vertices in $O(n^\omega \log n)$ time, where ω is the matrix multiplication exponent. Abboud et at. [1] extended this result to all-pairs λ-reachability by presenting an algorithm that takes $O((\lambda \log n)4^{\lambda+o(\lambda)} \cdot n^\omega)$ time. One of the interesting open questions is if for any constants $\lambda, k \geqslant 1$, we can compute an oracle that given any query vertex-pair $x, y \in V$ and any set F of k edge failures, reports (x, y)-λ-reachability in $\mathcal{G} \setminus F$ efficiently.

For any vertex $x \in V$, let $\mathcal{H}_x$ denote a (λ, k)-FT-BFP of $\mathcal{G}$ with x as the source. Our data structure simply stores the graph family $\{\mathcal{H}_x \mid x \in V\}$. Given any query vertex-pair (x, y) and any set F of k edges, we compute the (x, y)-max-flow in $\mathcal{H}_x$ by employing the max-flow algorithm of Chen et al. [10]. The time to compute the max-flow is $O(|E(\mathcal{H}_x)|^{1+o(1)})$, which is just $O(2^k\lambda n^{1+o(1)})$. Note that the total space used is bounded by $O(2^k\lambda n^2)$. Therefore, we have the following theorem.

▶ **Theorem 24.** *Given any directed graph $\mathcal{G} = (V, E)$ on n vertices, and any positive constants $\lambda, k \geqslant 1$, we can preprocess G in polynomial time to build an $O(n^2)$ size data structure that, given any query vertex-pair (x, y) and any set F of k edges, can determine the (x, y)-λ-reachability in $\mathcal{G} \setminus F$ in $O(n^{1+o(1)})$ time.*

6.2 FT-BFPs for graphs with non-unit capacities

We have shown till now that for any digraph $\mathcal{G}$ with unit capacities, one can compute a (λ, k)-FT-BFP with $O(2^k\lambda n)$ edges. We shall now show how to extend this result to a digraph with integer edge capacities such that flow values up to λ are preserved under bounded capacity decrement.

Let us first formalize the notion of FT-BFP under capacity decrement function.

▶ **Definition 25.** *Let $\mathcal{G} = (V, E, c)$ be a directed flow graph such that capacity of any edge is a positive integer, and let $s \in V$ be a designated source vertex. A subgraph $\mathcal{H} = (V, E_0 \subseteq E)$ of $\mathcal{G}$ is said to be a (λ, k)-Fault-Tolerant Bounded-Flow-Preserver if for any capacity decrement function $I : E(G) \to \mathbb{N}$ satisfying $\sum_{e \in E(G)} I(e) \leqslant k$, the following holds for the capacity function c^* defined as $c^*(e) = c(e) - I(e)$, for $e \in E$:*
For every $t \in V$,

$$\textsc{max-flow}(s, t, \mathcal{H}|c^*) = \begin{cases} \textsc{max-flow}(s, t, \mathcal{G}|c^*) & \text{if } \textsc{max-flow}(s, t, \mathcal{G}|c^*) \leqslant \lambda, \\ \text{At least } \lambda, & \text{otherwise;} \end{cases}$$

where, $\mathcal{H}|c^*$ and $\mathcal{G}|c^*$ are respectively the graphs $\mathcal{H}$ and $\mathcal{G}$ with capacity function c^*.

Let us now discuss the construction of (λ, k)-FT-BFPs. Let $\mathcal{G} = (V, E, c)$ be a digraph with integer edge capacities. We first transform $\mathcal{G}$ into a multigraph $\mathcal{G}^*$ by replacing an edge (x, y) of capacity $c(x, y)$ by exactly $c(x, y)$ copies of edge (x, y) of unit-capacity. Thus, for vertex $v \in V$, the s to v max-flow in graphs $\mathcal{G}$ and $\mathcal{G}^*$ are identical.

Now, let $\mathcal{H}^*$ be a (λ, k)-FT-BFP of multigraph $\mathcal{G}^*$. Then, a (λ, k)-FT-BFP of $\mathcal{G}$, say $\mathcal{H} = (V, E_0, c)$, can be obtained by simply retaining all those edges whose multiplicity in $\mathcal{H}^*$ is non-zero. The graph $\mathcal{H}$ will indeed be a (λ, k)-FT-BFP of $\mathcal{G}$ since a bounded capacity decrement in $\mathcal{G}$ corresponds to k-edge failures in $\mathcal{G}^*$.

References

1 Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019*, volume 132 of *LIPIcs*, pages 7:1–7:15, 2019. doi:10.4230/LIPICS.ICALP.2019.7.

2 Surender Baswana, Koustav Bhanja, and Abhyuday Pandey. Minimum+1 (s, t)-cuts and dual edge sensitivity oracle. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 15:1–15:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.15.

3 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: Breaking the o(m) barrier. *SIAM J. Comput.*, 48(4):1335–1363, 2019. doi:10.1137/17M114306X.

4 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault-tolerant subgraph for single-source reachability: General and optimal. *SIAM Journal on Computing*, 47(1):80–95, 2018. doi:10.1137/16M1087643.

5 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/S00453-012-9621-Y.

6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 18:1–18:14, 2016. doi:10.4230/LIPICS.STACS.2016.18.

7 Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013. doi:10.1016/J.IC.2012.10.009.

8 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 435–444, 2009. doi:10.1145/1536414.1536475.

9 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. *f*-sensitivity distance oracles and routing schemes. In *18th Annual European Symposium on Algorithms - ESA (1)*, pages 84–96, 2010. doi:10.1007/978-3-642-15775-2_8.

10 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

11 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 130:1–130:13, 2016. doi:10.4230/LIPICS.ICALP.2016.130.

12 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

13 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011*, pages 169–178, 2011. doi:10.1145/1993806.1993830.

14 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 506–515, 2009. doi:10.1137/1.9781611973068.56.

15 D. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 2010.

16 Loukas Georgiadis, Daniel Graf, Giuseppe F. Italiano, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs 2-reachability in o(nˆw log n) time. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 74:1–74:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.74.

17 Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. A brief note on single source fault tolerant reachability, 2019. arXiv:1904.08150.

18 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. In Anupam Gupta, Klaus Jansen, José Rolim, and Rocco Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 276–287, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32512-0_24.

19 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

20 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 481–490, 2015. doi:10.1145/2767386.2767408.

21 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, pages 779–790, 2013. doi:10.1007/978-3-642-40450-4_66.

22 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1073–1092, 2014. doi:10.1137/1.9781611973402.80.

23 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435, 2019. doi:10.1109/FOCS.2019.00034.

Optimal Sensitivity Oracle for Steiner Mincut

Koustav Bhanja
Department of CSE, IIT Kanpur, India

Abstract

Let $G = (V, E)$ be an undirected weighted graph on $n = |V|$ vertices and $S \subseteq V$ be a Steiner set. Steiner mincut is a well-studied concept, which also provides a generalization to both (s,t)-mincut (when $|S| = 2$) and global mincut (when $|S| = n$). Here, we address the problem of designing a compact data structure that can efficiently report a Steiner mincut and its capacity after the failure of any edge in G; such a data structure is known as a *Sensitivity Oracle* for Steiner mincut.

In the area of minimum cuts, although many Sensitivity Oracles have been designed in unweighted graphs, however, in weighted graphs, Sensitivity Oracles exist only for (s,t)-mincut [Annals of Operations Research 1991, NETWORKS 2019, ICALP 2024], which is just a special case of Steiner mincut. Here, we generalize this result from $|S| = 2$ to any arbitrary set $S \subseteq V$, that is, $2 \leq |S| \leq n$.

We first design an $\mathcal{O}(n^2)$ space Sensitivity Oracle for Steiner mincut by suitably generalizing the approach used for (s,t)-mincuts [Annals of Operations Research 1991, NETWORKS 2019]. However, the main question that arises quite naturally is the following.

Can we design a Sensitivity Oracle for Steiner mincut that breaks the $\mathcal{O}(n^2)$ bound on space?

In this article, we present the following two results that provide an answer to this question.

1. **Sensitivity Oracle:** Assuming the capacity of every edge is known,
   a. there is an $\mathcal{O}(n)$ space data structure that can report the capacity of Steiner mincut in $\mathcal{O}(1)$ time and
   b. there is an $\mathcal{O}(n(n - |S| + 1))$ space data structure that can report a Steiner mincut in $\mathcal{O}(n)$ time

   after the failure of any edge in G.
2. **Lower Bound:** We show that any data structure that, after the failure of any edge in G, can report a Steiner mincut or its capacity must occupy $\Omega(n^2)$ bits of space in the worst case, irrespective of the size of the Steiner set.

The lower bound in (2) shows that the assumption in (1) is essential to break the $\Omega(n^2)$ lower bound on space. Sensitivity Oracle in (1.b) occupies only subquadratic, that is $\mathcal{O}(n^{1+\epsilon})$, space if $|S| = n - n^{\epsilon} + 1$, for every $\epsilon \in [0, 1)$. For $|S| = n - k$ for any constant $k \geq 0$, it occupies only $\mathcal{O}(n)$ space. So, we also present the first Sensitivity Oracle occupying $\mathcal{O}(n)$ space for global mincut. In addition, we are able to match the existing best-known bounds on both space and query time for (s,t)-mincut [Annals of Operations Research 1991, NETWORKS 2019] in undirected graphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Network flows

Keywords and phrases mincut, (s, t)-mincut, Steiner mincut, fault tolerant structures, data structure, vital edges, vitality, sensitivity oracle

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.10

Related Version *Full Version*: https://arxiv.org/abs/2409.17715

Funding This research work is partially funded by Research-I Foundation of the Department of CSE, IIT Kanpur, India.

Acknowledgements I am grateful to my doctoral advisor Prof. Surender Baswana for reviewing this article and providing valuable feedback on improving its readability. I also want to thank an anonymous reviewer for his/her insightful comments on this article.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 10; pp. 10:1–10:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the real world, networks (graphs) are often subject to the failure of edges and vertices due to a variety of factors, such as physical damage, interference, or other disruptions. This can lead to changes in the solution to several graph problems. While these failures can happen at any location in the network at any time, they are typically short-lived. Naturally, it requires us to have compact data structures that can efficiently report the solution to the given graph problem (without computing from scratch) once any failure has occurred. Such data structures are known as *Sensitivity Oracles* for several graph problems. There exist elegant Sensitivity Oracles for many fundamental graph problems, such as shortest paths [6, 10], reachability [22, 13], traversals [25, 5], etc.

The minimum cut of a graph is also a fundamental concept of graph theory. Moreover, it has a variety of practical applications in the real world [1]. Designing Sensitivity Oracles for various minimum cuts of a graph has been an emerging field of research for the past few decades [4, 12, 19, 17, 7, 9, 8, 3]. There are two well-known mincuts of a graph. They are global mincut and (s,t)-mincut. Here, we design the first Sensitivity Oracle for global mincut in undirected weighted graphs that can handle the failure of any edge. The concept of Steiner mincut is also well-studied in the area of minimum cuts [18, 16, 19, 8, 14, 21, 23]; moreover, it has global mincut, as well as (s,t)-mincut, as just a special corner case. In this article, as our main result, we present the first *Sensitivity Oracle for Steiner mincut* for handling the failure of any edge in undirected weighted graphs. Interestingly, our result bridges the gap between the two extreme scenarios of Steiner mincut while matching their bounds, namely, (s,t)-mincut [2, 12] and global mincut (designed in this article). In addition, it also provides the first generalization from unweighted graphs [8, 18, 16, 19] to weighted graphs.

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges with non-negative real values assigned as the capacity to edges. We denote the capacity of an edge e by $w(e)$. Let $S \subseteq V$ be a *Steiner set* of G such that $|S| \geq 2$. A vertex s is called a *Steiner vertex* if $s \in S$; otherwise, s is called a *nonSteiner vertex*.

▶ **Definition 1** (Steiner cut). *A nonempty set $C \subset V$ is said to be a Steiner cut if there is at least one pair of Steiner vertices s, s' such that $s \in C$ and $s' \notin C$.*

For $S = V$, a Steiner cut is a *(global) cut*. Similarly, for $S = \{s, t\}$, a Steiner cut is an *(s,t)-cut*. A cut C is said to *separate* a pair of vertices u, v if $u \in C$ and $v \in \overline{C} = V \setminus C$ or vice versa. An edge $e = (u, v)$ is said to *contribute* to a cut C if C separates endpoints u, v of e. The *capacity of a cut C*, denoted by $c(C)$, is the sum of capacities of all contributing edges of C. A Steiner cut of the least capacity is known as the *Steiner mincut*, denoted by S-mincut. Let λ_S be the capacity of S-mincut. The problem of designing a Sensitivity Oracle for S-mincut for handling the failure of any edge is defined as follows.

▶ **Definition 2** (single edge Sensitivity Oracle for Steiner mincut). *For any graph G, a single edge Sensitivity Oracle for Steiner mincut is a compact data structure that can efficiently report a Steiner mincut and its capacity after the failure of any edge in G.*

For unweighed graphs, there exist single edge Sensitivity Oracles for global mincut [15], (s,t)-mincut [26, 4], and Steiner mincut [18, 16, 8]. Unfortunately, for weighted graphs, in the area of minimum cuts, the only existing results are single edge Sensitivity Oracles for (s,t)-mincut [2, 12, 3]. For undirected weighted graphs, Ausiello et al. [2], exploiting the Ancestor tree data structure of Cheng and Hu [12], designed the first single edge Sensitivity Oracle for (s,t)-mincut. Their Sensitivity Oracle occupies $\mathcal{O}(n^2)$ space. After the failure of any edge, it can report an (s,t)-mincut C and its capacity in $\mathcal{O}(|C|)$ and $\mathcal{O}(1)$ time, respectively. Recently, Baswana and Bhanja [3] complemented this result by showing that $\Omega(n^2 \log n)$ bits of space is required in the worst case, irrespective of the query time.

For Steiner mincuts, it follows from the above discussion that the existing Sensitivity Oracles are either for undirected unweighted graphs or only for a special case, when $|S| = 2$, in weighted graphs. Therefore, to provide a generalization of these results to any Steiner set, the following is an important question to raise.

Does there exist a single edge Sensitivity Oracle for S-mincut in undirected weighted graphs?

We show that the approach taken by Ausiello et al. [2] can be generalized from $S = \{s,t\}$ to any set $S \subseteq V$. This answers the above-mentioned question in the affirmative and leads to the following result.

▶ **Theorem 3.** *For any undirected weighted graph G on $n = |V|$ vertices, for every Steiner set S, there exists an $\mathcal{O}(n^2)$ space data structure that, after the failure of any edge in G, can report an S-mincut C and its capacity in $\mathcal{O}(|C|)$ time and $\mathcal{O}(1)$ time respectively.*

The space and query time of the Sensitivity Oracle in Theorem 3 match with the existing optimal results for (s,t)-mincut [12, 2, 3]. The lower bound of $\Omega(n^2 \log n)$ bits of space in [3] is only for $|S| = 2$. To the best of our knowledge, no lower bound is known for any $|S| > 2$. Therefore, the main question that we address in this article arises quite naturally as follows.

▶ **Question 1.** *For undirected weighted graphs, does there exist a single edge Sensitivity Oracle for S-mincut that breaks the quadratic bound on space and still achieves optimal query time if $|S| > 2$?*

1.1 Our Results

A Sensitivity Oracle in a weighted graph addresses queries in a more generic way [3]. Given any edge e and any value Δ satisfying $\Delta \geq 0$, the aim is to efficiently report the solution of a given problem after reducing the capacity of edge e by Δ. In this generic setting, using the well-known Gomory and Hu Tree data structure [20], we design the first single edge Sensitivity Oracle for global mincut in weighted graphs that achieves optimal query time.

▶ **Theorem 4** (Sensitivity Oracle for Global Mincut). *For any undirected weighted graph $G = (V,E)$ on $n = |V|$ vertices, there is an $\mathcal{O}(n)$ space data structure that, given any edge e in G and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report the capacity of global mincut in $\mathcal{O}(1)$ time and a global mincut C in $\mathcal{O}(|C|)$ time after reducing the capacity of edge e by Δ.*

The result in Theorem 4 matches the bounds on both space and query time with the best-known single edge Sensitivity Oracles for global mincut in unweighted graphs [15].

Now, in order to bridge the gap between the two extreme scenarios of Steiner set ($|S| = n$ and $|S| = 2$) while matching their bounds, we present our main result that breaks the $\mathcal{O}(n^2)$ space bound of Theorem 3, and answers Question 1 in the affirmative.

▶ **Theorem 5** (Sensitivity Oracle for Steiner Mincut). *Let $G = (V,E)$ be an undirected weighted graph on $n = |V|$ vertices and $m = |E|$ edges. For any Steiner set S of G,*

1. *there is an $\mathcal{O}(n)$ space rooted tree $\mathcal{T}(G)$ that, given any edge $e \in E$ and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report the capacity of S-mincut in $\mathcal{O}(1)$ time after reducing the capacity of edge e by Δ and*

2. *there is an $\mathcal{O}(n(n-|S|+1))$ space data structure $\mathcal{F}(G)$ that, given any edge $e \in E$ and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report an S-mincut C in $\mathcal{O}(|C|)$ time after reducing the capacity of edge e by Δ.*

For any $\epsilon \in [0,1)$, the space occupied by the single edge Sensitivity Oracle for S-mincut in Theorem 5(2) is subquadratic, that is $\mathcal{O}(n^{1+\epsilon})$, for $|S| = n - n^{\epsilon} + 1$. Moreover, it approaches to $\mathcal{O}(n)$ as $|S|$ tends to n. In particular, for $|S| = n-k$, for any constant $k \geq 0$, it occupies only $\mathcal{O}(n)$ space.

Observe that our results in Theorem 5 interestingly match the bounds on both space and query time for the two extreme scenarios of the Steiner set. On one extreme ($|S| = n$), it occupies $\mathcal{O}(n)$ space for global mincut. On the other extreme ($|S| = 2$), it occupies $\mathcal{O}(n^2)$ space, which match the best-known existing results for (s,t)-mincut [2, 12, 3]. Finally, the time taken by our Sensitivity Oracle to answer any query is also worst-case optimal.

We also provide lower bounds on both space and query time of Sensitivity Oracles for S-mincut. Our first lower bound is for reporting the capacity of S-mincut and our second lower bound is for reporting an S-mincut.

▶ **Theorem 6** (Lower Bound for Reporting Capacity). *Let D be any data structure that can report the capacity of Steiner mincut after the failure of any edge for undirected weighted graphs on n vertices. Data structure D must occupy $\Omega(n^2 \log n)$ bits of space in the worst case, irrespective of the query time and the size of the Steiner set.*

For reporting the capacity of S-mincut, Theorem 6 provides a generalization of the existing lower bound on both space and time for (s,t)-mincut by Baswana and Bhanja [3]. However, for reporting an S-mincut, no lower bound on space or query time for single edge Sensitivity Oracle was known till date, even for the two extreme scenarios of Steiner set. So, the following theorem is the first lower bound for reporting an S-mincut after the failure of any edge.

▶ **Theorem 7** (Lower Bound for Reporting Cut). *Let D be any data structure that can report a Steiner mincut C in $\mathcal{O}(|C|)$ time after the failure of any edge for undirected weighted graphs on n vertices. Data structure D must occupy $\Omega(n^2)$ bits of space in the worst case, irrespective of the size of the Steiner set.*

▶ Remark 8. It is assumed in Theorem 5 that the query edge e is present in G and the change in capacity (that is, Δ) provided with the query is at most $w(e)$. So, the lower bounds of $\Omega(n^2)$ bits of space in Theorem 6 and Theorem 7 do not violate the sub-quadratic space data structures in Theorem 5. Moreover, the assumption in Theorem 5 seems practically justified. This is because, as discussed in [3], in the real world, the capacity of an edge reduces only if the edge actually exists in the graph, and furthermore, it can reduce by a value at most the capacity of the edge.

1.2 Related Works

In the seminal works by Dinitz and Vainshtein [18, 16, 19], they designed an $\mathcal{O}(\min\{n\lambda_S, m\})$ space data structure, known as *Connectivity Carcass*, for storing all S-mincuts of an unweighted undirected graph. It can report an S-mincut in $\mathcal{O}(m)$ time and its capacity in $\mathcal{O}(1)$ time. Baswana and Pandey [8], using Connectivity Carcass as the foundation, designed an $\mathcal{O}(n)$ space single edge Sensitivity Oracle for S-mincut in undirected unweighted graphs that also reports an S-mincut in $\mathcal{O}(n)$ time. Their result matches the bounds on both space and time for the existing result on the two extreme scenarios of S-mincut, namely, (s,t)-mincut [26] and global mincut [15]. The result on S-mincut in [8] also acts as the foundation of single edge Sensitivity Oracles for all-pairs mincut [8]

For directed weighted graphs, Baswana and Bhanja [3] presented a single edge Sensitivity Oracle for (s,t)-mincut that matches both space and query time of the undirected weighted graph results [12, 2].
Providing a generalization from the two extreme scenarios of the Steiner set ($|S| = n$ and $|S| = 2$) is also addressed for various problems, namely, computing Steiner mincut [18, 19, 14, 21, 23], Steiner connectivity augmentation and splitting-off [11], construction of a cactus graph for Steiner mincuts [19, 21].

1.3 Organization of the Article

This article is organized as follows. Section 2 contains the basic preliminaries. We first construct an $\mathcal{O}(n^2)$ space single edge Sensitivity Oracle for Steiner mincut in Section 3. In Section 4, we design an $\mathcal{O}(n)$ space single edge Sensitivity Oracle for reporting only the capacity of Steiner mincut. A linear space single edge Sensitivity Oracle for global mincut is designed in Section 5. Our main result on the subquadratic space single edge Sensitivity Oracle for Steiner mincut is developed in Section 6. Finally, we conclude in Section 7. The proofs of the lower bounds are provided in the full version of this article.

2 Preliminaries

In this section, we define a set of basic notations and properties of cuts. Let $G \setminus \{e\}$ denote the graph obtained from G after the removal of edge e. We now define the concept of crossing cuts, introduced by Dinitz, Karzanov, and Lomonosov [15].

▶ **Definition 9** (crossing cuts). *A pair of cuts C and C' in G is said to be crossing if each of the four sets $C \cap C'$, $C \setminus C'$, $C' \setminus C$, and $\overline{C \cup C'}$ is nonempty.*

The following concept of mincut for an edge and vital edges are to be used crucially in the construction of our data structure.

▶ **Definition 10** (Mincut for an edge). *A Steiner cut C is said to be a mincut for an edge e if e contributes to C and $c(C) \leq c(C')$ for every Steiner cut C' in which e contributes.*

▶ **Definition 11** (Vital Edge). *Let e be an edge and C be a mincut for edge e. Edge e is said to be a vital edge if its removal reduces the capacity of Steiner mincut, that is, $c(C) - w(e) < \lambda_S$.*

We now define a *special* mincut for an edge.

▶ **Definition 12** (Nearest mincut for an edge). *A mincut C for an edge $e = (x, y) \in E$ with $x \in C$ is said to be a nearest mincut for e if there is no mincut C' for e such that $x \in C'$ and $C' \subset C$. The set of all nearest mincuts for an edge e is denoted by $N(e)$.*

▶ **Lemma 13** (Sub-modularity of Cuts (Problem 48(a,b) in [24])). *For any two sets $A, B \subset V$,*

1. *$c(A) + c(B) \geq c(A \cap B) + c(A \cup B)$ and*
2. *$c(A) + c(B) \geq c(A \setminus B) + c(B \setminus A)$.*

▶ **Definition 14** (Laminar family of cuts). *A set of cuts $\mathcal{L}$ is said to form a laminar family if, for any pair of cuts $C_1, C_2 \in \mathcal{L}$, exactly one of the three is true – $C_1 \cap C_2$ is an empty set, $C_1 \subseteq C_2$, and $C_2 \subseteq C_1$.*

A rooted tree $\mathcal{T}_{\mathcal{L}}$ representing a laminar family $\mathcal{L}$. For any given laminar family $\mathcal{L}$ of cuts in G, we can construct an $\mathcal{O}(n)$ space rooted tree $\mathcal{T}_{\mathcal{L}}$ that stores every cut belonging to $\mathcal{L}$ as follows. Every vertex x of G is mapped to a unique node in $\mathcal{T}_{\mathcal{L}}$, denoted by $\phi_{\mathcal{L}}(x)$. Every node μ in $\mathcal{T}_{\mathcal{L}}$ represents a unique cut C in $\mathcal{L}$ as follows. Cut C is the set of vertices mapped to the subtree rooted at μ (including μ). For any pair of nodes μ and ν in $\mathcal{T}_{\mathcal{L}}$, μ is a child of ν if and only if the cut represented by μ is a maximal proper subset of the cut represented by ν. The minimal cuts of $\mathcal{L}$ are represented by leaf nodes of $\mathcal{T}_{\mathcal{L}}$. For any vertex x in G, let $\textsc{SubTree}(x)$ denote the set of all vertices mapped to the subtree rooted at $\phi_{\mathcal{L}}(x)$ (including $\phi_{\mathcal{L}}(x)$) in $\mathcal{T}_{\mathcal{L}}$. This leads to the following lemma.

▶ **Lemma 15.** *For any laminar family $\mathcal{L}$ of cuts in G, there exists an $\mathcal{O}(n)$ space rooted tree $\mathcal{T}_{\mathcal{L}}$ such that a cut $C \in \mathcal{L}$ if and only if there exists a node μ (except root node) of $\mathcal{T}_{\mathcal{L}}$ and C is the set of vertices mapped to the subtree rooted at μ (including node μ).*

3 An $\mathcal{O}(n^2)$ Space Sensitivity Oracle for Steiner Mincut

In this section, we first provide the limitations of the previous results in unweighted graphs. Later, we design an $\mathcal{O}(n^2)$ space single edge Sensitivity Oracle for S-mincut.

Limitations of the existing results. For unweighted graphs, the following property is used crucially to design every existing single edge Sensitivity Oracle.
Property P_1: *Failure of an edge e reduces the capacity of S-mincut if and only if edge e contributes to an S-mincut.*
Dinitz and Vainshtein [18, 16, 19] designed the following quotient graph, known as the flesh graph, of G. Flesh graph is obtained by contracting every pair of vertices in G that are not separated by any S-mincut. The construction ensures that every pair of vertices in flesh is separated by an S-mincut of G. Every vertex in G is mapped to a unique vertex in flesh. Therefore, the endpoints of any edge e are mapped to different vertices in flesh if and only if failure of e reduces capacity of S-mincut. Thus, by Property P_1, storing the $\mathcal{O}(n)$ space mapping of vertices of G to the vertices of flesh is sufficient to answer the capacity of S-mincut in $\mathcal{O}(1)$ time after the failure of any edge. In addition, flesh graph can be used to report an S-mincut after the failure of any edge in G in $\mathcal{O}(m)$ time.
Flesh graph is one of the three components of Connectivity Carcass designed by Dinitz and Vainshtein [18, 16, 19]; the other two components are *Skeleton* and *Projection mapping*. Recently, Baswana and Pandey [8], exploiting the properties of all the three components of Connectivity Carcass established an *ordering* among the vertices of flesh graph. By using Property P_1, they showed that this ordering, along with Skeleton and Projection mapping, can be used to design an $\mathcal{O}(n)$ space single edge Sensitivity Oracle for S-mincut in unweighted graphs. This single edge Sensitivity Oracle can report an S-mincut in $\mathcal{O}(n)$ time.

Unfortunately, for weighted graphs, it is easy to observe that multiple edges can exist that do not contribute to any S-mincut but failure of each of them reduces the capacity of S-mincut. Hence, in weighted graphs, Property P_1 no longer holds. So, the existing structures are not suitable for handling the failure of weighted edges. It requires us to explore the structure of mincuts for every edge whose both endpoints belong to the same node of flesh graph. Moreover, the capacity of mincut for these edges can be quite *large* compared to the capacity of S-mincut.

Sensitivity Oracle for Steiner Mincut: $\mathcal{O}(n^2)$ Space

We now give a proof of Theorem 3 by designing an $\mathcal{O}(n^2)$ space single edge Sensitivity Oracle for S-mincut. Let F be any arbitrary real-valued function defined on cuts. Cheng and Hu [12] presented the following result. There is an $\mathcal{O}(n^2)$ space data structure, known as Ancestor tree, that, given any pair of vertices u and v, reports a cut C of the least capacity (F-value) separating u, v in $\mathcal{O}(|C|)$ time and the capacity of C in $\mathcal{O}(1)$ time.

In order to design Ancestor tree for Steiner cuts, similar to (s,t)-mincuts given by Ausiello et al. [2], we define function F for Steiner cuts as follows.

$$\text{For a set } C \subset V,\ F(C) = \begin{cases} c(C), & \text{if } C \text{ is a Steiner cut} \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

Let $e = (x, y)$ be any failed edge. Ancestor tree can report a cut C of the least capacity separating x and y in $\mathcal{O}(|C|)$ time and its capacity in $\mathcal{O}(1)$ time. By Equation 1, C is also a Steiner cut separating x and y. Therefore, by Definition 10, C is a mincut for edge e. Hence, the new capacity of S-mincut is either $c(C) - w(e)$ or remains λ_S if $c(C) - w(e) \geq \lambda_S$. By storing the capacities of all edges of G, we can determine whether $c(C) - w(e) < \lambda_S$ in $\mathcal{O}(1)$ time. If the capacity of S-mincut reduces, then we can report C in $\mathcal{O}(|C|)$ time; otherwise report an S-mincut C_m in $\mathcal{O}(|C_m|)$ time. This completes the proof of Theorem 3.

4 A Sensitivity Oracle for Reporting Capacity of Steiner Mincut

In this section, we address the problem of reporting the capacity of S-mincut after reducing the capacity of an edge $e \in E$ by a value Δ satisfying $0 \leq \Delta \leq w(e)$. We denote this query by $\textsc{cap}(e, \Delta)$. Observe that a trivial data structure for answering query $\textsc{cap}$ occupies $\mathcal{O}(m)$ space if we store the capacity of mincut for each vital edge in G. For $|S| = 2$ in directed weighted graphs, Baswana and Bhanja [3] designed an $\mathcal{O}(n)$ space data structure that implicitly stores all vital edges for (s,t)-mincut and the capacity of their mincuts. We extend their approach from vital edges to any set of edges in undirected weighted graphs in order to establish the following. For any Steiner set S, with $2 \leq |S| \leq n$, there exists an $\mathcal{O}(n)$ space data structure that can answer query $\textsc{cap}$ in $\mathcal{O}(1)$ time.

Let $\mathcal{E} \subseteq E$ and $V(\mathcal{E})$ denote the smallest set of vertices such that, for each edge $(u, v) \in \mathcal{E}$, both u and v belongs to $V(\mathcal{E})$. We first design an $\mathcal{O}(|V(\mathcal{E})|)$ space rooted full binary tree for answering query $\textsc{cap}$ for all edges in $\mathcal{E}$. In Section 6, this construction also helps in designing a compact structure for reporting mincuts for a *special* subset of edges. Later in this section, we show an extension to $\mathcal{O}(n)$ space rooted full binary tree for answering query $\textsc{cap}$ for all edges in E.

Let $C(e)$ denote a mincut for an edge e. Note that $C(e)$ is a Steiner cut as well (Definition 10). We say that an edge e belongs to a set $U \subset V$ if both endpoints of e belong to U. Suppose $C(e)$ is a mincut for an edge e belonging to $V(\mathcal{E})$ such that, for every other edge $e' \in V(\mathcal{E})$, $c(C(e)) \leq c(C(e'))$. Let e' be an edge from $V(\mathcal{E})$. If e' contributes to $C(e)$, it follows from the selection of edge e that $C(e)$ is a Steiner cut of the least capacity to which e' contributes. Hence, $C(e)$ is a mincut for edge e' as well. This ensures that $C(e)$ partitions the set of all edges belonging to $\overline{V(\mathcal{E})}$ into three sets – edges of $V(\mathcal{E})$ belonging to $C(e) \cap V(\mathcal{E})$, edges of $V(\mathcal{E})$ belonging to $\overline{C(e)} \cap V(\mathcal{E})$, and edges of $V(\mathcal{E})$ that contribute to $C(e)$. This leads to a recursive procedure (Algorithm 1) for the construction of a tree $\mathcal{T}$. Each internal node μ of tree $\mathcal{T}$ has three fields – (i) μ.cap stores the capacity of mincut for the selected edge at μ, (ii) $\mu.\textsc{left}$ points to the left child of μ, and (iii) $\mu.\textsc{right}$ points to the right child of μ. Each vertex $u \in V(\mathcal{E})$ is mapped to a leaf node of $\mathcal{T}$, denoted by $\mathbb{L}(u)$. We invoke Algorithm 1 with $U = V(\mathcal{E})$.

Algorithm 1 Construction of Tree $\mathcal{T}$.

```
procedure SteinerTreeConstruction($U$)
    Create a node $\nu$;
    For any set $U \subseteq V$, let $E(U)$ denote the edges whose both endpoints belong to $U$;
    if there is no edge in $E(U)$ then
        for each vertex $x \in U$ do $\mathbb{L}(x) \leftarrow \nu$;
        end for
    else
        Select an edge $e \in E(U)$ such that $c(C(e)) \leq c(C(e'))$, $\forall$ edge $e' \in E(U)$;
        Assign $\nu$.cap $\leftarrow c(C(e))$;
        $\nu$.LEFT $\leftarrow$ SteinerTreeConstruction($U \cap C(e)$);
        $\nu$.RIGHT $\leftarrow$ SteinerTreeConstruction($U \cap \overline{C(e)}$);
    end if
    return $\nu$;
end procedure
```

Observe that tree $\mathcal{T}$ resulting from Algorithm 1 is a full binary tree. There are $\mathcal{O}(|V(\mathcal{E})|)$ leaf nodes. So, the space occupied by the tree is $\mathcal{O}(|V(\mathcal{E})|)$.

Answering Query cap($e = (x, y), \Delta$). Suppose edge e belongs to $\mathcal{E}$. Let μ be the lowest common ancestor (LCA) of $\mathbb{L}(x)$ and $\mathbb{L}(y)$ in $\mathcal{T}$. It follows from the construction of tree $\mathcal{T}$ that field μ.cap at node μ in $\mathcal{T}$ stores the capacity of mincut for edge e. Therefore, if $\mu.\text{cap} - \Delta < \lambda_S$, then we report μ.cap as the new capacity of S-mincut; otherwise, the capacity of S-mincut does not change. It leads to the following lemma.

▶ **Lemma 16.** *Let $G = (V, E)$ be an undirected weighted graph on $n = |V|$ vertices. For any Steiner set $S \subseteq V$ and a set of edges $\mathcal{E} \subseteq E$, there is an $\mathcal{O}(|V(\mathcal{E})|)$ space full binary tree $\mathcal{T}_{\mathcal{E}}$ that, given any edge $e \in \mathcal{E}$ and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report the capacity of S-mincut in $\mathcal{O}(1)$ time after reducing the capacity of edge e by Δ.*

We now answer query CAP(e, Δ) where edge $e \in E$. Observe that edge e in query CAP can be either a vital or a nonvital edge. In order to determine whether an edge is vital or not, we design a full binary tree $\mathcal{T}_E$ by invoking Algorithm 1 with $U = V$ since $\mathcal{E} = E$. Let us denote the tree by $\mathcal{T}(G)$. By Lemma 16, the size of tree $\mathcal{T}(G)$ is $\mathcal{O}(n)$. It is now easy to observe that an edge e is a vital edge in graph G if and only if the capacity of the Steiner mincut in graph $G \setminus \{e\}$ is $\mu.\text{cap} - w(e) < \lambda_S$, where node μ is the LCA$(\mathbb{L}(x), \mathbb{L}(y))$. This leads to the following lemma.

▶ **Lemma 17.** *Let $G = (V, E)$ be an undirected weighted graph on $n = |V|$ vertices. For any Steiner set $S \subseteq V$, there is an $\mathcal{O}(n)$ space full binary tree $\mathcal{T}(G)$ that, given any edge $e \in E$ and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report the capacity of S-mincut in $\mathcal{O}(1)$ time after reducing the capacity of edge e by Δ.*

Lemma 17 completes the proof of Theorem 5(1).

▶ Remark 18. Ancestor tree of Cheng and Hu [12] can also be used to design an $\mathcal{O}(n)$ space data structure for answering query CAP in $\mathcal{O}(1)$ time. However, Ancestor tree alone does not seem sufficient to establish Lemma 16, which is used crucially to achieve the subquadratic space single edge Sensitivity Oracle for S-mincut stated in Theorem 5.

5 An $\mathcal{O}(n)$ Space Sensitivity Oracle for Global Mincut

The well-known (s,t)-mincut is one extreme scenario of S-mincut when $|S|=2$. In weighted graphs, designing a single edge Sensitivity Oracle for (s,t)-mincut has been addressed quite extensively [2, 12, 3]. Moreover, each of them occupies $\mathcal{O}(n^2)$ space. However, to this day, no nontrivial single edge Sensitivity Oracle exists for global mincut, which is the other extreme scenario of S-mincut when $|S|=n$. We now present the first single edge Sensitivity Oracle for global mincut that occupies only $\mathcal{O}(n)$ space and achieves optimal query time.

Let λ_V be the capacity of global mincut. Given any edge e, we want to determine the capacity of mincut for edge e for Steiner set $S=V$. Observe that, for $S=V$, every cut in the graph is a Steiner cut (or global cut). Exploiting this insight, we can state the following interesting relation between global mincut and all-pairs mincuts (or (u,v)-mincut, for every $u,v\in V$).

▶ **Lemma 19.** *For an edge (u,v), C is a cut of the least capacity that separates u,v if and only if C is a mincut for edge (u,v).*

Gomory and Hu [20] designed the following tree data structure for all-pairs mincuts, which is widely known as Gomory Hu Tree.

▶ **Theorem 20** (Gomory Hu Tree [20])**.** *For any undirected weighted graph $G=(V,E)$ on $n=|V|$ vertices, there is an $\mathcal{O}(n)$ space undirected weighted tree $\mathcal{T}_{GH}$ on vertex set V that satisfies the following property. Let u,v be any pair of vertices in G. A cut of the least capacity separating u,v in $\mathcal{T}_{GH}$ is also a cut of the least capacity separating u,v in G. Moreover, $\mathcal{T}_{GH}$ can report a cut C of the least capacity separating u,v in $\mathcal{O}(|C|)$ time and its capacity in $\mathcal{O}(1)$ time.*

Let $\mathcal{T}_{GH}$ be a Gomory Hu Tree of G. By Theorem 20, for every pair of vertices u,v in G, $\mathcal{T}_{GH}$ stores a cut of the least capacity separating u,v. By Lemma 19, it follows that, for $S=V$, $\mathcal{T}_{GH}$ stores a mincut for every edge in G. Hence, it acts as a single edge Sensitivity Oracle for global mincut and can report a mincut C for any given edge e in $\mathcal{O}(|C|)$ time and its capacity in $\mathcal{O}(1)$ time. Therefore, after reducing $w(e)$ by a value Δ, if $c(C)-\Delta<\lambda_V$, we can report a global mincut and its capacity optimally using $\mathcal{O}(n)$ space. This establishes the results in Theorem 4.

Now, for both extreme scenarios of Steiner mincuts, we have a single edge Sensitivity Oracle. Interestingly, the Sensitivity Oracle for global mincut achieves better than quadratic space. The question that arises is how to generalize these results to any Steiner set.

6 A Sensitivity Oracle for Steiner Mincut: Breaking Quadratic Bound

In this section, we address the problem of reporting an S-mincut after reducing the capacity of any given edge $e\in E$ by any given value Δ satisfying $0<\Delta\le w(e)$. We denote this query by $\textsc{cut}(e,\Delta)$. Our objective is to design a data structure that breaks $\mathcal{O}(n^2)$ bound on space for efficiently answering query $\textsc{cut}$. A simple data structure can be designed by augmenting tree $\mathcal{T}(G)$ in Theorem 5(1) as follows. For each internal node μ of tree $\mathcal{T}(G)$, Algorithm 1 selects an edge e in Step 7 and stores the capacity of mincut $C(e)$ for edge e in μ.cap. Observe that if we augment node μ with $C(e)$, then it helps in answering query $\textsc{cut}$ as well. However, the augmented tree occupies $\mathcal{O}(n^2)$ space, which defeats our objective.

For global mincut ($S = V$), observe that Gomory Hu Tree essentially acts as a data structure that stores at least one mincut for every edge quite compactly. To design a more compact data structure for answering query cut for S-mincut compared to Theorem 3, we take an approach of designing a data structure that can compactly store at least one mincut for every edge.

We begin by a *classification* of all edges of graph G. This classification not only helps in combining the approaches taken for (s,t)-mincut and global mincut but also provides a way to design a compact data structure for efficiently answering query cut for any Steiner set S.

A Classification of All Edges. An edge e_1 in G belongs to

- Type-1 if both endpoints of e_1 belong to $V \setminus S$.
- Type-2 if both endpoints of e_1 belong to S.
- Type-3 if exactly one endpoint of e_1 belongs to S.

Given any edge e_1, we can classify e_1 into one of the above-mentioned three types in $\mathcal{O}(1)$ time using sets V and S. We can take care of edges from Type-1 by extending an approach for (s,t)-mincut given by Baswana and Bhanja [3]. Similarly, we can handle edges from Type-2 by extending the approach used for global mincut (Theorem 4). However, the **main challenge** arises in designing a data structure for compactly storing a mincut for all edges from Type-3. We now design a compact data structure for efficiently answering query cut for each type of edges separately.

6.1 An $\mathcal{O}((n - |S|)n)$ Space Data Structure for All Edges from Type-1

We aim to design a data structure for answering query cut for all edges from Type-1. Each edge from Type-1 has both endpoints in set $V \setminus S$. The number of vertices in $V \setminus S$ is $n - |S|$. Therefore, trivially storing a mincut for every edge would occupy $\mathcal{O}((n-|S|)^2 n)$ space, which is $\mathcal{O}(n^3)$ for $|S| = k$ for any constant $k \geq 2$. Exploiting the fact that the number of distinct endpoints of all edges from Type-1 is at most $n - |S|$, we design an $\mathcal{O}(n(n - |S|))$ space data structure for all edges from Type-1 using Algorithm 1 and Lemma 16 as follows.

Let E_1 be the set of all edges from Type-1. It follows from Lemma 16 that, using Algorithm 1, it is possible to design a rooted full binary tree $\mathcal{T}_{E_1}$ occupying $\mathcal{O}(n - |S|)$ space for answering query cap(e, Δ) when edge e is from Type-1. We augment each internal node μ of $\mathcal{T}_{E_1}$ with a mincut for the edge selected by Algorithm 1 (in Step 7) while processing node μ. The resulting structure occupies $\mathcal{O}((n - |S|)n)$ space and acts as a data structure for answering query cut for all edges from Type-1. Hence the following lemma holds.

▶ **Lemma 21** (Sensitivity Oracle for Type-1 Edges). *For any Steiner set $S \subseteq V$, there is an $\mathcal{O}((n - |S|)n)$ space data structure that, given any edge e from Type-1 and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report an S-mincut C in $\mathcal{O}(|C|)$ time after reducing the capacity of edge e by Δ.*

6.2 An $\mathcal{O}(n)$ Space Data Structure for All Edges from Type-2

In this section, we design a data structure for answering query cut for all edges from Type-2. For each edge from Type-2, both endpoints are Steiner vertices. So, the number of distinct endpoints of edges from Type-2 can be at most $|S|$. Trivially, storing a mincut for every edge from Type-2 would occupy $\mathcal{O}(|S|^2 n)$ space, which is $\mathcal{O}(n^3)$ if $|S| = \mathcal{O}(n)$. In a similar way as designing $\mathcal{T}_{E_1}$ for all edges from Type-1 (Lemma 21), by using Lemma 16 and Algorithm 1, it is possible to design an $\mathcal{O}(|S|n)$ space data structure for answering query cut for all edges from Type-2. Unfortunately, it defeats our objective because, for $|S| = n$ or global mincuts, it occupies $\mathcal{O}(n^2)$ space. Interestingly, by exploiting the fact that both the endpoints of every edge from Type-2 are Steiner vertices, we are able to show that a Gomory Hu Tree of graph G is sufficient for answering query cut for all edges from Type-2.

▶ **Lemma 22.** *Let $\mathcal{T}_{GH}$ be a Gomory Hu Tree of G. Then, for any edge $e = (u, v)$ from Type-2, a cut of the least capacity separating u and v in $\mathcal{T}_{GH}$ is a mincut for edge e in G.*

Proof. Let $\mathcal{T}_{GH}$ be a Gomory Hu Tree of G. By Theorem 20, $\mathcal{T}_{GH}$ stores a cut of the least capacity separating every pair of vertices in G. Let (u, v) be any edge from Type-2. Suppose C is a cut of the least capacity separating u and v in $\mathcal{T}_{GH}$. So, edge (u, v) is contributing to C in G. By definition of Type-2 edges, both u and v are Steiner vertices. Therefore, C is a Steiner cut of G in which edge (u, v) is contributing. It follows from Theorem 20 that C is also a cut of the least capacity in G that separates u and v. So, C is also a Steiner cut of the least capacity in which edge (u, v) is contributing in G. Hence, C is a mincut for (u, v). ◀

Let $e = (u, v)$ be any edge from Type-2 and $\mathcal{T}_{GH}$ be a Gomory Hu Tree of G. By Theorem 20, $\mathcal{T}_{GH}$ can report in $\mathcal{O}(|C|)$ time a cut C of the least capacity in $\mathcal{T}_{GH}$ that separates u and v. By Lemma 22, C is a mincut for edge e in G. This establishes the following lemma.

▶ **Lemma 23** (Sensitivity Oracle for Type-2 Edges). *For any Steiner set $S \subseteq V$, there is an $\mathcal{O}(n)$ space data structure that, given any edge e from Type-2 and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report an S-mincut C in $\mathcal{O}(|C|)$ time after reducing the capacity of edge e by Δ.*

For global mincut or $S = V$, both endpoints of every edge are Steiner vertices. Therefore, Theorem 4 can also be seen as a corollary of Lemma 23.

6.3 An $\mathcal{O}((n - |S|)n)$ Space Data Structure for All Edges from Type-3

The objective is to design a data structure for answering query cut for all edges from Type-3. Observe that the size of the smallest set of vertices that contains all the endpoints of all edges from Type-3 can be $\Omega(n)$. Therefore, using Lemma 16 and Algorithm 1, we can have an $\mathcal{O}(n^2)$ space data structure, which is no way better than the trivial data structure for answering query cut (Theorem 3). Now, each edge from Type-3 has exactly one nonSteiner endpoint. So, unlike edges from Type-2, Lemma 22 no longer holds for edges from Type-3. This shows the limitations of the approaches taken so far in designing a data structure for answering query cut. Trivially storing a mincut for every edge from Type-3 requires $\mathcal{O}((n - |S|)n|S|)$ space. For $|S| = \frac{n}{k}$, any constant $k \geq 2$, it occupies $\mathcal{O}(n^3)$ space. Interestingly, we present a data structure occupying only $\mathcal{O}((n - |S|)n)$ space for answering query cut for all edges from Type-3.

For any edge $e_1 = (x, u)$ from Type-3 with $x \in S$, without loss of generality, we assume that any mincut C for edge e_1 contains the Steiner vertex x, otherwise consider $\overline{C}$. Note that the set of global mincuts and (s, t)-mincuts are closed under both intersection and union. This property was crucially exploited in designing a compact structure for storing them [15, 26]. To design a compact structure for storing a mincut for every edge from Type-3, we also explore the relation between a pair of mincuts for edges from Type-3. Let A and B be mincuts for edges e_1 and e_2 from Type-3, respectively. Unfortunately, it turns out that if A crosses B, then it is quite possible that neither $A \cap B$ nor $A \cup B$ is a mincut for e_1 or e_2 even if both are Steiner cuts (refer to Figure 1(i)). This shows that mincuts for edges from Type-3 are not closed under intersection or union. To overcome this hurdle, we first present a partitioning of the set of edges from Type-3 based on the nonSteiner vertices as follows.

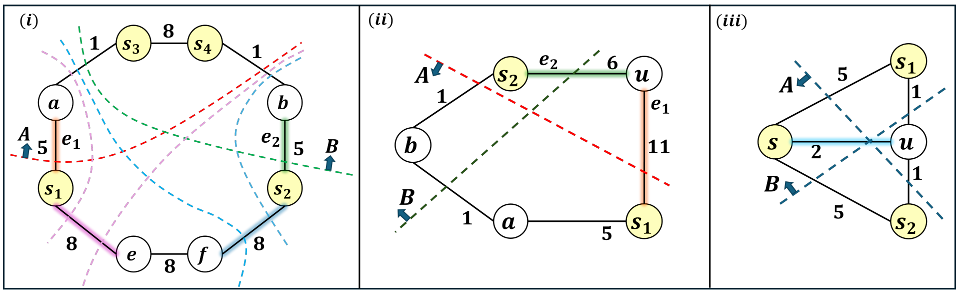


Figure 1 Yellow vertices are Steiner vertices. A mincut for an edge is represented by the same color. (i) Mincuts A, B are for edges $e_1 = (s_1, a)$ and $e_2 = (s_2, b)$. Observe that $A \cap B$ and $A \cup B$ are Steiner cuts but not mincuts for edges e_1, e_2. Moreover, cuts $A \setminus B$ and $B \setminus A$, in which edges e_1 and e_2 are contributing, are not even Steiner cuts. (ii) Edges e_1 and e_2 are vital and from Type-3(u). Mincuts A and B for edges e_1 and e_2 are crossing, but $A \cap B$ and $A \cup B$ are not Steiner cuts. (iii) Edge (s, u) is from Type-3 and $N((s, u)) = \{A, B\}$.

Let $V' \subseteq V \setminus S$ be the smallest set of nonSteiner vertices such that every edge from Type-3 has endpoint in V'. Let u be any vertex from V'. Let Type-3(u) be the set that contains all edges from Type-3 having u as one of the two endpoints. We aim to design an $\mathcal{O}(n)$ space data structure that can report a mincut for each edge from Type-3(u). This is because storing an $\mathcal{O}(n)$ space data structure for every nonSteiner vertex of G would lead to an $\mathcal{O}((n - |S|)n)$ space data structure.

In order to design an $\mathcal{O}(n)$ space data structure for edges from Type-3(u), we consider the set of nearest mincuts (Definition 12) for edges from Type-3(u). The following lemma provides a strong reason behind the use of nearest mincuts for edges from Type-3(u).

▶ **Lemma 24** (Disjoint Property). *Let $C \in N(e_1)$ and $C' \in N(e_2)$ such that $e_1 = (x, u)$, $e_2 = (x', u)$ are edges from Type-3(u). Then, $x' \notin C$ and $x \notin C'$ if and only if $C \cap C' = \emptyset$.*

Proof. Suppose $x' \notin C$ and $x \notin C'$. Assume to the contrary that $C \cap C' \neq \emptyset$. Observe that $x \in C \setminus C'$ and $x' \in C' \setminus C$. As a result, $C \setminus C'$ as well as $C' \setminus C$ is a Steiner cut. It is given that C is the nearest mincut for edge (x, u) and $x \in C \setminus C'$. This implies that $c(C \setminus C') > c(C)$. It follows from sub-modularity of cuts (Lemma 13(2)) that $c(C' \setminus C) < c(C')$. Therefore, we get a Steiner cut $C' \setminus C$ of capacity strictly less than $c(C')$ and edge (x', u) is a contributing edge of Steiner cut $C' \setminus C$, a contradiction.

The proof of the converse part is immediate. ◀

Let $e_1 = (x, u)$ and $e_2 = (x', u)$ be any pair of edges from Type-3(u). Let C be a nearest mincut for e_1 and C' be a nearest mincut for e_2. Lemma 24 essentially states that if e_2 contributes to $C' \setminus C$ and e_1 contributes to $C \setminus C'$, then C must not cross C'. Now, the problem arises when one of the two edges e_1, e_2 is contributing to a nearest mincut for the other edge. Firstly, there might exist multiple nearest mincuts for an edge (refer to Figure 1(iii)). It seems quite possible that an edge, say e_2, is contributing to one nearest mincut for e_1 and is not contributing to another nearest mincut for e_1. Secondly, the union of a pair of nearest mincuts for an edge e_1 from Type-3(u) might not even be a Steiner cut if they cross (refer to Figure 1(iii)). Hence, the union of them can have a capacity strictly less than the capacity of mincut for e. So, it seems that the nearest mincuts for edges from Type-3(u) appear quite *arbitrarily*. It might not be possible to have an $\mathcal{O}(n)$ space structure for storing them. Interestingly, we are able to circumvent all the above challenges as follows.

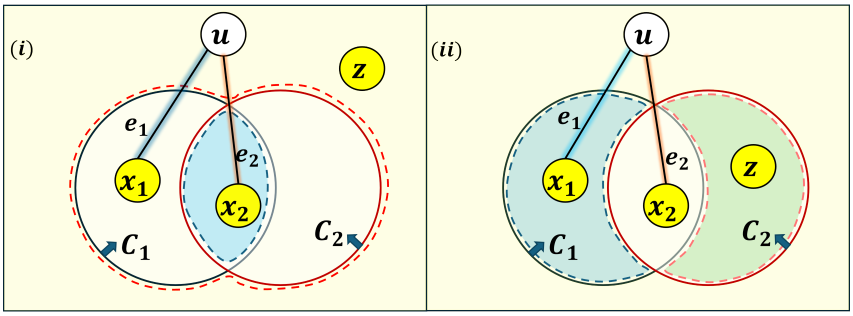


Figure 2 Illustration of the proof of Lemma 25. (i) There is a Steiner vertex z in $\overline{C_1 \cup C_2}$. The red dashed cut shows cut $C_1 \cup C_2$ and the blue dashed cut (blue region) shows cut $C_1 \cap C_2$. (ii) There is a Steiner vertex z in $C_2 \setminus C_1$. Blue dashed cut (blue region) is $C_1 \setminus C_2$. Similarly, red dashed cut (light green region) is $C_2 \setminus C_1$.

Observe that we are interested in only those edges from Type-3(u) whose failure reduces S-mincut. They are the set of all vital edges that belong to Type-3(u), denoted by VitType-3(u). By exploiting *vitality* of edges from VitType-3(u), we establish the following crucial result for any pair of crossing mincuts for edges from VitType-3(u). Interestingly, this result holds even if the union of a pair of mincuts for a pair of edges from VitType-3(u) is not always a Steiner cut (refer to Figure 1(ii)).

▶ **Lemma 25** (Property of Intersection). *Let C_1 and C_2 be mincuts for edges $e_1 = (x_1, u)$ and $e_2 = (x_2, u)$ from VitType-3(u) respectively. Steiner vertex x_2 is present in C_1 if and only if $C_1 \cap C_2$ is a mincut for edge e_2.*

Proof. Suppose Steiner vertex x_2 is present in C_1. Since u is a common endpoint of both edges e_1, e_2, we have $u \notin C_1 \cup C_2$. C_1 is a mincut for edge e_1, so, $x_1 \in C_1$. Now, there are two possibilities – either (1) $x_1 \notin C_2$ or (2) $x_1 \in C_2$. We now establish each case separately.

Case 1. Suppose $x_1 \notin C_2$, in other words, $x_1 \in C_1 \setminus C_2$. It implies that $C_1 \setminus C_2$ is nonempty. Observe that C_2 is not a subset of C_1; otherwise, $C_1 \cap C_2 = C_2$ is a mincut for e_2 and the lemma holds. So, $C_2 \setminus C_1$ is also nonempty. Let $c(C_1) = \lambda_1$ and $c(C_2) = \lambda_2$. Since $x_2 \in C_1 \cap C_2$, edge e_2 is contributing to C_1. Therefore, $c(C_2) \leq c(C_1)$; otherwise, C_2 is not a mincut for edge e_2. Since C_1 is a Steiner cut, there must be a Steiner vertex z such that $z \notin C_1$. Based on the position of z with respect to cut C_2, observe that z appears either (1.1) in $\overline{C_1 \cup C_2}$ or (1.2) in $C_2 \setminus C_1$ (refer to Figure 2).

Case 1.1. Suppose $z \in \overline{C_1 \cup C_2}$ (refer to Figure 2(i)). Observe that $C_1 \cup C_2$ is a Steiner cut in which edge e_1 is contributing. So, the capacity of $C_1 \cup C_2$ has to be at least λ_1. By sub-modularity of cuts (Lemma 13(1)), $c(C_1 \cap C_2) + c(C_1 \cup C_2) \leq \lambda_1 + \lambda_2$. It follows that $c(C_1 \cap C_2) \leq \lambda_2$. Since $C_1 \cap C_2$ is a Steiner cut in which edge e_2 is contributing, therefore, the capacity of $C_1 \cap C_2$ is exactly λ_2. Hence $C_1 \cap C_2$ is a mincut for edge e_2.

Case 1.2. We show that this case does not arise. Assume to the contrary that $z \in C_2 \setminus C_1$ (refer to Figure 2(ii)). Here, we crucially exploit the fact that edge e_2 is a vital edge from Type-3(u). Let us now consider graph $G \setminus \{e_2\}$. Since, in graph G, edge e_2 is a vital edge and $c(C_2) \leq c(C_1)$, therefore, in graph $G \setminus \{e_2\}$, the capacity of S-mincut is $\lambda_2 - w(e_2)$ and C_2 is an S-mincut. In G, edge e_2 is also a contributing edge of C_1. Therefore, the capacity of C_1 in $G \setminus \{e_2\}$ is $\lambda_1 - w(e_2)$. Without causing any ambiguity, let us denote the capacity of any cut A in $G \setminus \{e_2\}$ by $c(A)$. By sub-modularity of cuts (Lemma 13(2)), in graph $G \setminus \{e_2\}$, we have $c(C_1 \setminus C_2) + c(C_2 \setminus C_1) \leq \lambda_1 + \lambda_2 - 2w(e_2)$. Recall that $x_1 \in C_1 \setminus C_2$ and $z \in C_2 \setminus C_1$. So, both $C_1 \setminus C_2$ and $C_2 \setminus C_1$ are Steiner cuts in graph $G \setminus \{e_2\}$. Therefore, the capacity of $C_2 \setminus C_1$ is at least $\lambda_2 - w(e_2)$. It follows that the capacity of $C_1 \setminus C_2$ in $G \setminus \{e_2\}$ is at most $\lambda_1 - w(e_2)$. We now obtain graph G by adding edge e_2 to graph $G \setminus \{e_2\}$. Observe that edge e_2 does not contribute to Steiner cut $C_1 \setminus C_2$. Therefore, the capacity of cut $C_1 \setminus C_2$ remains the same in graph G, which is at most $\lambda_1 - w(e_2)$. Since e_2 is a vital edge, so, $w(e_2) > 0$. This implies that we have $\lambda_1 - w(e_2) < \lambda_1$. Therefore, for cut $C_1 \setminus C_2$ in G, $C_1 \setminus C_2$ is a Steiner cut and has a capacity that is strictly less than λ_1. Moreover, edge e_1 is contributing to $C_1 \setminus C_2$. So, C_1 is not a mincut for edge e_1, a contradiction.

Case 2. In this case, we have $x_1 \in C_2$. Since C_1 and C_2 both are Steiner cuts, observe that either (2.1) there is at least one Steiner vertex z in $\overline{C_1 \cup C_2}$ or (2.2) there exists a pair of Steiner vertices z_1, z_2 such that $z_1 \in C_1 \setminus C_2$ and $z_2 \in C_2 \setminus C_1$. The proof of case (2.1) is along similar lines to the proof of case (1.1). So, let us consider case (2.2). Edges e_1 and e_2 are contributing to both C_1 and C_2. It implies that $c(C_1) = c(C_2)$. Let $c(C_1)$ (or $c(C_2)$) be λ. Let us consider graph $G \setminus \{e_2\}$. Since e_2 is a vital edge, the capacity of S-mincut is $\lambda - w(e_2)$. The capacity of cuts C_1 and C_2 in $G \setminus \{e_2\}$ is $\lambda - w(e_2)$ since e_2 contributes to both of them. Without causing any ambiguity, let us denote the capacity of any cut A in $G \setminus \{e_2\}$ by $c(A)$. By sub-modularity of cuts (Lemma 13(2)), $c(C_1 \setminus C_2) + c(C_2 \setminus C_1) \leq 2\lambda - 2w(e_2)$. Now, it is given that $z_1 \in C_1 \setminus C_2$ and $z_2 \in C_2 \setminus C_1$. Therefore, in $G \setminus \{e_2\}$, $C_1 \setminus C_2$ and $C_2 \setminus C_1$ are Steiner cuts. It follows that $c(C_1 \setminus C_2)$, as well as $c(C_2 \setminus C_1)$, is exactly $\lambda - w(e_2)$. Let us obtain graph G from $G \setminus \{e_2\}$. Observe that e_2 contributes neither to $C_1 \setminus C_2$ nor to $C_2 \setminus C_1$. Therefore, the capacity of cuts $C_1 \setminus C_2$ and $C_2 \setminus C_1$ remains the same in G, which is $\lambda - w(e_2)$. Since e_2 is vital, $w(e_2) > 0$. So, $\lambda - w(e_2) < \lambda_S < \lambda$, where λ_S is the capacity of S-mincut in G. Hence, we have a Steiner cut of capacity strictly smaller than S-mincut, a contradiction.

We now prove the converse part. Suppose $C_1 \cap C_2$ is a mincut for edge e_2. Since x_2 is a Steiner vertex, by Definition 10, x_2 is in $C_1 \cap C_2$. Since $C_1 \cap C_2 \subseteq C_1$, x_2 is in C_1. This completes the proof. ◀

For any pair of nonSteiner vertices $a, b \in V'$, it turns out that Lemma 25 does not necessarily hold as follows. As shown in Figure 1(i), s_2 is present in nearest mincut A of edge (s_1, a) but nearest mincut B for edge (s_2, b) crosses A.

Recall that our objective is to design an $\mathcal{O}(n)$ space structure for storing a mincut for every edge from VitType-3(u). By Lemma 24, the set of nearest mincuts for all edges from VitType-3(u) satisfies Disjoint property. By exploiting Lemma 25, we now establish two interesting properties satisfied by the nearest mincuts for all edges from VitType-3(u) – Uniqueness Property (Lemma 26) and Subset Property (Lemma 27). These properties help in designing an $\mathcal{O}(n)$ space data structure for storing them. We first establish the uniqueness property in the following lemma.

▶ **Lemma 26** (Uniqueness Property). *For any edge $e = (x, u)$ from VitType-3(u), the nearest mincut for edge e is unique.*

Proof. Suppose C_1 and C_2 are a pair of distinct nearest mincuts for edge e. It follows from Lemma 25 that $C = C_1 \cap C_2$ is a mincut for edge e. So, C is a proper subset of both C_1 and C_2, which contradicts that C_1 and C_2 are nearest mincuts for edge e. ◀

Although the nearest mincut for each edge from VitType-3(u) is unique (Lemma 26), the Uniqueness Property alone can only guarantee a data structure occupying $\mathcal{O}(n|S|)$ space for all edges from VitType-3(u). To achieve a better space, we now explore the relation between the nearest mincuts for a pair of edges from VitType-3(u). Since nearest mincut for an edge from VitType-3(u) is unique (Lemma 26), without causing any ambiguity, we consider $N(e)$ to denote the unique nearest mincut for an edge e from VitType-3(u).

Let $e_1 = (x_1, u)$ and $e_2 = (x_2, u)$ be a pair of edges from VitType-3(u). If neither $x_1 \in N(e_2)$ nor $x_2 \in N(e_1)$, then, by Lemma 24, $N(e_1)$ is disjoint from $N(e_2)$. The other cases are when $x_1 \in N(e_2)$ or $x_2 \in N(e_1)$. We exploit Lemma 25 to establish the following property. This property states that $N(e_1)$ is either identical to $N(e_2)$ or one of $\{N(e_1), N(e_2)\}$ contains the other.

▶ **Lemma 27** (Subset Property). *Let (x, u) and (x', u) be a pair of edges from VitType-3(u). Then, $x' \in N((x, u))$ if and only if $N((x', u)) \subseteq N((x, u))$.*

Proof. Let $C = N((x, u))$ and $C' = N((x', u))$. Let us assume to the contrary that $C' \nsubseteq C$. It is given that $x' \in C$. Therefore, by Lemma 25, $C \cap C'$ is also a mincut for edge (x', u). This contradicts that C' is a nearest mincut for edge (x', u).

Since $N((x', u)) \subseteq N((x, u))$ and (x', u) is a contributing edge of $N((x', u))$ with $x' \in N((x', u))$, therefore, x' also belong to $N((x, u))$. This completes the proof. ◀

Let (x_1, u) and (x_2, u) be edges from VitType-3(u), where $x_1, x_2 \in S$. Let C_1 and C_2 be nearest mincuts for edges (x_1, u) and (x_2, u), respectively. It follows from Lemma 27 and Lemma 24 that there are three possibilities for C_1 and C_2 – C_1 is the same as C_2, one of C_1 and C_2 is a proper subset of the other, and C_1 is disjoint from C_2. Therefore, for any vertex $u \in V'$, the set containing the nearest mincuts for every edge from VitType-3(u) forms a Laminar family $\mathcal{L}(u)$ (Definition 14) on set V. This inference, along with Lemma 15, leads to the following result.

▶ **Lemma 28.** *There is an $\mathcal{O}(n)$ space tree $\mathcal{T}_{\mathcal{L}(u)}$ that satisfies the following property. For each edge (x, u) from VitType-3(u) with $x \in S$, SubTree(x) of tree $\mathcal{T}_{\mathcal{L}(u)}$ is the nearest mincut for edge (x, u).*

Data Structure $\mathcal{F}_3$ for all vital edges from Type-3. For each nonSteiner vertex $u \in V'$, we construct a tree $\mathcal{T}_{\mathcal{L}(u)}$ based on the laminar family $\mathcal{L}(u)$ consisting of the nearest mincuts for all edges from VitType-3(u). Since V' contains nonSteiner vertices of G only, there can be at most $n - |S|$ vertices in V'. Therefore, by Lemma 28, the overall space occupied by the data structure is $\mathcal{O}(n(n - |S|))$.

Reporting a mincut for a vital edge from Type-3 using $\mathcal{F}_3$. Given any vital edge $e = (x, u)$ from Type-3, where $x \in S$ and $u \in V \setminus S$, by following Lemma 28, we report the set of vertices stored in SubTree(x) of tree $\mathcal{T}_{\mathcal{L}(u)}$ as the nearest mincut for edge (x, u).

Note that given any edge e from Type-3 and any value Δ satisfying $0 \leq \Delta \leq w(e)$, by using the data structure of Lemma 17, we can determine in $\mathcal{O}(1)$ time whether the capacity of S-mincut reduces after reducing $w(e)$ by Δ. This leads to the following lemma for answering query cut for all edges from Type-3.

▶ **Lemma 29** (Sensitivity Oracle for Type-3 Edges). *For any Steiner set $S \subseteq V$, there is an $\mathcal{O}((n - |S|)n)$ space data structure that, given any edge e from Type-3 and any value Δ satisfying $0 \leq \Delta \leq w(e)$, can report an S-mincut C in $\mathcal{O}(|C|)$ time after reducing the capacity of edge e by Δ.*

Algorithm 2 Answering Query CUT.

```
 1: procedure CUT($e = (x, y), \Delta$)
 2:     Let $C$ be a Steiner mincut of $G$;
 3:     Assign MINCUT ← $C$;
 4:     TYPE ← the type of edge $e$ determined using the endpoints $\{x, y\}$;
 5:     if TYPE == 1 then
 6:         Assign MINCUT ← a mincut for edge $e$ using data structure of Lemma 21;
 7:     else if TYPE == 2 then
 8:         Assign MINCUT ← a mincut for edge $e$ using data structure of Lemma 23;
 9:     else if TYPE == 3 then
10:         Verify using the data structure in Lemma 17 whether $e$ is vital;
11:         if $e$ is a vital edge then
12:             Assign MINCUT ← a mincut for edge $e$ using data structure of Lemma 29;
13:         else
14:             do nothing;
15:         end if
16:     end if
17:     return MINCUT;
18: end procedure
```

Lemma 21, Lemma 23, and Lemma 29 complete the proof of Theorem 5(2).

The pseudo-code for answering query CUT is provided in Algorithm 2. Algorithm 2 is invoked with the failed edge e and the change in capacity Δ of edge e satisfying $0 \leq \Delta \leq w(e)$. In Step 10 of Algorithm 2, the change in capacity (Δ) is required to determine if edge e is a vital edge. Otherwise, Algorithm 2 fails to report the valid S-mincut after reducing the capacity of edge e.

7 Conclusion

We have designed the first Sensitivity Oracle for Steiner mincuts in weighted graphs. It also includes the first Sensitivity Oracle for global mincut in weighted graphs. Interestingly, our Sensitivity Oracle occupies space subquadratic in n when $|S|$ approaches n and also achieves optimal query time. On the other hand, it matches the bounds on both space and query time with the existing best-known results for (s, t)-mincut [12, 2].

Our quadratic space single edge Sensitivity Oracle does not assume that the capacity of the failed edge is known. We have also complemented this result with matching lower bounds. Now, it would be great to see whether there is any single edge Sensitivity Oracle for Steiner mincut that occupies only $\mathcal{O}(n)$ space assuming the capacity of the failed edge is known.

Finally, our obtained structure that breaks the quadratic bound is quite simple as it is a forest of $\mathcal{O}(n - |S|)$ trees. We strongly believe that our techniques and structures will be quite useful for addressing several problems in the future, including the problem of designing a Sensitivity Oracle for S-mincut that can handle failure of multiple edges.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

2 Giorgio Ausiello, Paolo Giulio Franciosa, Isabella Lari, and Andrea Ribichini. Max flow vitality in general and st-planar graphs. *Networks*, 74(1):70–78, 2019. doi:10.1002/net.21878.

3 Surender Baswana and Koustav Bhanja. Vital edges for (s, t)-mincut: Efficient algorithms, compact structures, & optimal sensitivity oracles. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.17.

4 Surender Baswana, Koustav Bhanja, and Abhyuday Pandey. Minimum+1 (s, t)-cuts and dual-edge sensitivity oracle. *ACM Trans. Algorithms*, 19(4):38:1–38:41, 2023. doi:10.1145/3623271.

5 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic dfs in undirected graphs: Breaking the o(m) barrier. *SIAM Journal on Computing*, 48(4):1335–1363, 2019. doi:10.1137/17M114306X.

6 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single-source fault tolerant shortest path. *ACM Transactions on Algorithms (TALG)*, 16(4):1–22, 2020. doi:10.1145/3397532.

7 Surender Baswana, Shiv Gupta, and Till Knollmann. Mincut sensitivity data structures for the insertion of an edge. *Algorithmica*, 84(9):2702–2734, 2022. doi:10.1007/S00453-022-00978-0.

8 Surender Baswana and Abhyuday Pandey. Sensitivity oracles for all-pairs mincuts. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 581–609. SIAM, 2022. doi:10.1137/1.9781611977073.27.

9 Koustav Bhanja. Minimum+ 1 steiner cuts and dual edge sensitivity oracle: Bridging the gap between global cut and (s, t)-cut. *arXiv preprint*, 2024. doi:10.48550/arXiv.2406.15129.

10 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1396–1409, 2023. doi:10.1145/3564246.3585251.

11 Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2449–2488. SIAM, 2023. doi:10.1137/1.9781611977554.CH95.

12 Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.*, 33(3):199–213, 1991. doi:10.1007/BF02115755.

13 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2016.

14 Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 167–176, 2003. doi:10.1145/780542.780568.

15 Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. On the structure of the system of minimum edge cuts in a graph. *Issledovaniya po Diskretnoi Optimizatsii*, pages 290–306, 1976.

16 Ye Dinitz and Alek Vainshtein. Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 302–311, 1995.

17 Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+ 1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 509–518, 1995. doi:10.1145/225058.225268.

18 Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 716–725, 1994. doi:10.1145/195058.195442.

19 Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.

20 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

21 Zhongtian He, Shang-En Huang, and Thatchaphol Saranurak. Cactus representations in polylogarithmic max-flow via maximal isolating mincuts. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1465–1502. SIAM, 2024. doi:10.1137/1.9781611977912.60.

22 Giuseppe F Italiano, Adam Karczmarz, and Nikos Parotsidis. Planar reachability under single vertex or edge failures. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2739–2758. SIAM, 2021. doi:10.1137/1.9781611976465.163.

23 Stephen Jue and Philip N. Klein. A near-linear time minimum steiner cut algorithm for planar graphs. *CoRR*, abs/1912.11103, 2019. arXiv:1912.11103.

24 L. Lovász. *Combinatorial problems and exercises.* North-Holland Publishing Co., Amsterdam-New York, 1979.

25 Merav Parter. Dual failure resilient BFS structure. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490. ACM, 2015. doi:10.1145/2767386.2767408.

26 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. doi:10.1007/BFb0120902.

Temporal Queries for Dynamic Temporal Forests

Davide Bilò
Department of Information Engineering, Computer Science, and Mathematics,
University of L'Aquila, Italy

Luciano Gualà
Department of Enterprise Engineering, University of Rome "Tor Vergata", Italy

Stefano Leucci
Department of Information Engineering, Computer Science, and Mathematics,
University of L'Aquila, Italy

Guido Proietti
Department of Information Engineering, Computer Science, and Mathematics,
University of L'Aquila, Italy

Alessandro Straziota
Department of Enterprise Engineering, University of Rome "Tor Vergata", Italy

Abstract

In a temporal forest each edge has an associated set of time labels that specify the time instants in which the edges are available. A temporal path from vertex u to vertex v in the forest is a selection of a label for each edge in the unique path from u to v, assuming it exists, such that the labels selected for any two consecutive edges are non-decreasing.

We design linear-size data structures that maintain a temporal forest of rooted trees under addition and deletion of both edge labels and singleton vertices, insertion of root-to-node edges, and removal of edges with no labels. Such data structures can answer temporal reachability, earliest arrival, and latest departure queries. All queries and updates are handled in polylogarithmic worst-case time. Our results can be adapted to deal with latencies. More precisely, all the worst-case time bounds are asymptotically unaffected when latencies are uniform. For arbitrary latencies, the update time becomes amortized in the incremental case where only label additions and edge/singleton insertions are allowed as well as in the decremental case in which only label deletions and edge/singleton removals are allowed.

To the best of our knowledge, the only previously known data structure supporting temporal reachability queries is due to Brito, Albertini, Casteigts, and Travençolo [Social Network Analysis and Mining, 2021], which can handle general temporal graphs, answers queries in logarithmic time in the worst case, but requires an amortized update time that is quadratic in the number of vertices, up to polylogarithmic factors.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Dynamic graph algorithms

Keywords and phrases temporal graphs, temporal reachability, earliest arrival, latest departure, dynamic forests

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.11

Related Version *Full Version*: https://arxiv.org/abs/2409.18750

1 Introduction

Temporal graphs, a.k.a. time-varying graphs, are graphs in which each edge is only available in specific time instants. These graphs are crucial for modeling and analyzing a variety of complex systems such as social networks, communication networks, transportation systems, and biological networks, where relationships and interactions evolve.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 11; pp. 11:1–11:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A simple and widely-adopted model of temporal graphs is the one of Kempe, Kleinberg, and Kumar [19] in which a temporal graph is modeled as a graph G where each edge has an assigned integral temporal label representing a time instant in which the edge is available. Temporal paths are time-respecting walks on G, i.e., walks in which any two consecutive traversed edges have non-decreasing labels. This model naturally generalizes to the case in which an edge may be traversed in multiple time instants, i.e., when each edge of G is associated with a set of integer labels, and many other generalizations have also been proposed (see, e.g., [17]).

Due to their generality, many algorithmic problems regarding temporal graphs have been considered in the literature, including the computation of good temporal paths [1, 10, 22] w.r.t. several quality measures, sparsification [7, 8, 11, 12, 19], exploration [13, 14], a temporal version of the classical vertex cover problem [2, 15], and network creation games [6], to mention a few. Perhaps surprisingly, the problem of designing dynamic data structures maintaining information on temporal connectivity has only been addressed quite recently, despite the vast amount of literature for the non-temporal case (see, e.g., [16, 18] and the references therein). In this regard, Brito, Albertini, Casteigts, and Travençolo [9] provide a data structure that can answer *temporal reachability* queries in the *incremental* setting, i.e., under the addition of new edges or of new labels to already existing edges. This data structure has size $O(n^2\tau)$, and an amortized update time of $O(n^2 \log \tau)$, where n is the number of vertices of the temporal graph and τ is its *lifetime*, i.e., the number of distinct labels. Given two vertices u, v and two time instants t, t', it can report whether there exists a temporal path from u to v that departs from u no earlier than t and arrives in v no later than t'. The worst-case time required by such a query is $O(\log \tau)$ and the corresponding path can be retrieved in $O(\log \tau)$ worst-case time per edge.

In [4] the authors consider the problem of maintaining a directed graph in which edge (u, v) exists if there is a temporal path from u to v in G, and show how to efficiently update the *time-transitive closure* in the restricted *chronological incremental* case, where the inserted labels must be monotonically non decreasing.

Our results. We focus on the case in which the temporal graph of interest is acyclic, i.e., it is a temporal forest F, and we design data structures that support insertions and deletions of edge labels and can answer temporal reachability queries along with the more general *earliest arrival* (EA) and *latest departure* (LD) queries. Given two distinct vertices u, v and time instant t, an EA (resp. LD) query reports the smallest arrival time (reps. largest departure time) among those of all temporal paths from u to v that depart no earlier than t (resp. arrive no later than t).

Our main data structure is dynamic also w.r.t. a second aspect. In addition of being able to add and remove labels from the edges of F, it also supports the addition and deletion of singleton vertices along with *link* and *cut* operations in a temporal forest of rooted trees. The link operation adds a new edge (u, v) with label ℓ in F from the root u of any tree to any other vertex v in a different tree, thus merging the two trees into one. The cut operation deletes the last label of an edge e and removes e from F, thus splitting the tree T that contained e into two trees T_1, T_2, where the root of T_1 coincides with that of T and T_2 is rooted in the unique endvertex of e in T_2.

Our data structure has linear size, and can be updated in $O(\log M)$ worst-case time per operation, where M is the total number of (not necessarily distinct) labels in the forest at the time of the operation. EA and LD queries require $O(\log L \cdot \log M)$ worst-case time, where L is the maximum number of labels on a single edge at the time of the operation, and a temporal reachability query can be answered in $O(\log M)$ worst-case time (see Theorem 3).

In the special case of temporal paths with fixed topology (i.e., when only insertions and deletions of edge labels are allowed), we can improve the worst-case time complexity of EA and LD queries to $O(\log M)$, thus shaving a $O(\log L)$ factor (see Theorem 2).

Our data structure can also be adapted to the more general case of temporal forests with *latencies*, in which the labels on a generic edge are pairs (ℓ, d), with $d \geq 0$, encoding the fact that the edge can be traversed at time ℓ from any endvertex to reach the other endvertex at time $\ell+d$. The data structure retains the same worst-case asymptotic guarantees when latencies are *uniform* (see Corollary 7). For arbitrary latencies, we consider both the *incremental* and the *decremental* scenarios. In the incremental scenario we support the addition of new labels to existing edges, the addition of singleton vertices, and the above link operation, while the decremental scenario only involves label/singleton deletions and cut operations. In both cases, the above $O(\log M)$ upper bound on the update time becomes amortized (see Theorem 6).

2 Preliminaries

A *temporal forest* is an undirected forest $F = (V(F), E(F))$ paired with a function $\lambda : E(F) \to \mathscr{P}(\mathbb{Z})$ that associates a set $\lambda(e) \subseteq \mathbb{Z}$ of labels to each edge $e \in E(F)$. A *temporal path* π from vertex $u \in V(F)$ to vertex $v \in V(F) \setminus \{u\}$ in F is a sequence of pairs $\langle (e_1, \ell_1), (e_2, \ell_2), \dots, (e_k, \ell_k) \rangle$ such that $\langle e_1, e_2, \dots, e_k \rangle$ is a path from u to v in F, $\ell_i \in \lambda(e_i)$ for all $i = 1, \dots, k$, and $\ell_i \leq \ell_{i+1}$ for all $i = 1, \dots, k-1$. The departure time $\text{departure}(\pi)$ and the arrival time $\text{arrival}(\pi)$ of π are ℓ_1 and ℓ_k, respectively.

Given two distinct vertices $u, v \in V(F)$, and $t_a, t_d \in \mathbb{Z} \cup \{-\infty, +\infty\}$, $\Pi_F(u, v, t_d, t_a)$ is the set of all temporal paths from u to v in F having a departure time of at least t_d and an arrival time of at most t_a. For a time $t \in \mathbb{Z} \cup \{-\infty\}$, the *earliest arrival time* of a temporal path from u to v with departure time at least t is denoted by $\text{EA}(u, v, t)$ and is defined as $\min_{\pi \in \Pi(u,v,t,+\infty)} \text{arrival}(\pi)$. If no temporal path from u to v with departure time at least t exists in F, i.e., if $\Pi(u, v, t, +\infty) = \emptyset$, we define $\text{EA}(u, v, t) = +\infty$. A path $\pi \in \Pi(u, v, t, +\infty)$ such that $\text{arrival}(\pi) = \text{EA}(u, v, t)$ is called an *earliest arrival path*.

Similarly, we denote the *latest departure time* of a temporal path from u to v having an arrival time of at most $t \in \mathbb{Z} \cup \{+\infty\}$ as $\text{LD}(u, v, t) = \max_{\pi \in \Pi(u,v,-\infty,t)} \text{departure}(\pi)$. If $\Pi(u, v, -\infty, t) = \emptyset$, then we let $\text{LD}(u, v, t) = -\infty$. A path $\pi \in \Pi(u, v, t, +\infty)$ such that $\text{arrival}(\pi) = \text{LD}(u, v, t)$ is called a *latest departure path*. As an edge case, we define $\text{EA}(u, u, t) = \text{LD}(u, u, t) = t$ for all $u \in V(F)$.

We design a data structure for maintaining information on a dynamic temporal forest of rooted trees. The data structure must efficiently answer to the following queries:

Earliest arrival time (EA query): Given two vertices $u, v \in V(F)$ with $u \neq v$, and a time $t \in \mathbb{Z} \cup \{-\infty\}$, report $\text{EA}(u, v, t)$;

Latest departure time (LD query): Given two vertices $u, v \in V(F)$ with $u \neq v$, and a time $t \in \mathbb{Z} \cup \{+\infty\}$, report $\text{LD}(u, v, t)$;

Temporal reachability: Given two vertices $u, v \in V(F)$ with $u \neq v$, and two times $t_a, t_d \in \mathbb{Z} \in \{-\infty, +\infty\}$ with $t_a \leq t_d$, report whether there exists a temporal path π from u to v in F with $\text{departure}(\pi) \geq t_a$ and $\text{arrival}(\pi) \leq t_d$.

Clearly, a data structure that supports EA queries or LD queries also supports temporal reachability queries with the same worst-case query time. The update operations we consider are the following:

Label addition: Given an edge $e \in E(F)$ and a value $\ell \in \mathbb{Z} \setminus \lambda(e)$, add ℓ to $\lambda(e)$;
Label deletion: Given an edge $e \in E(F)$ with $|\lambda(e)| \geq 2$ and some $\ell \in \lambda(e)$, delete ℓ from $\lambda(e)$;
Link: Given two vertices u, v where u is the root of some tree T in F and v is some vertex of a tree T' of F with $T' \neq T$, and a label $\ell \in \mathbb{Z}$, add the edge (u, v) to F and set $\lambda((u, v)) = \{\ell\}$. The new tree resulting from merging T with T' retains the root of T';
Cut: Given an edge $e \in E(F)$ with $|\lambda(e)| = 1$, remove the edge e from the tree T of F containing e, thus splitting T into two new trees T_1, T_2 where T_1 retains the root of T and T_2 is rooted in the unique endvertex of e in T_2;
Singleton addition/deletion: Add a new (resp. remove an existing) singleton vertex to (resp. from) F.

If u, v, w are vertices of F such that w lies on the unique path between u and v in F we have that $\text{EA}(u, v, t) = \text{EA}(w, v, \text{EA}(u, w, t))$ and $\text{LD}(u, v, t) = \text{LD}(u, w, \text{LD}(w, v, t))$.

Let $-F$ be the temporal forest such that $V(F) = V(-F)$, $E(F) = E(-F)$, and for every $e \in E(F)$, the set of labels of e in $-F$ is $\{-\ell \mid \ell \in \lambda(e)\}$. One can observe that:

▶ **Lemma 1.** *The value of $LD(u, v, t)$ in F coincides with $-EA(v, u, -t)$ in $-F$.*

Finally, given a subset X of a universe possessing a strict total order, and an element x of the universe, we define the *successor* $\text{succ}(x, X)$ (resp. *predecessor* $\text{pred}(x, X)$) of x w.r.t. X as the minimum element $y \in X$ s.t. $y \geq x$ (resp. the maximum element $y \in X$ s.t. $y \leq x$). If such an element y does not exist, $\text{succ}(x, X) = +\infty$. Similarly, the *strict successor* (resp. *strict predecessor*) is defined as $\text{succ}(x, X \setminus \{x\})$ (resp. $\text{pred}(x, X \setminus \{x\})$).

3 Warm up: a data structure for temporal paths

As a warm up, in this section we consider the simpler case in which the forest F is actually a fixed path, and the only supported operations are label additions and deletions to/from the edges of F. We use this section to introduce some of the ideas of our construction, which we generalize to the case of dynamic temporal forests in Section 4 that also contains the correctness proofs. More precisely, we establish the following result:

▶ **Theorem 2.** *Given a temporal path with n vertices, it is possible to build in $O(n + M \log L)$ time, a data structure of linear size supporting EA and LD queries under label insertions and deletions in $O(\log M)$ worst-case time per operation, where L is the maximum number of labels on the same temporal edge, and M is the total number of (not necessarily distinct) labels in the path at the time of the operation.*

Fix an arbitrary endvertex v_0 of the path F, and let v_i be the unique vertex at hop-distance i from v_0 in F. We think of F as a tree rooted in v_{n-1}. With this interpretation, the main technical difficulty lies in designing a data structure capable of answering *upward* EA queries, i.e., EA queries from some vertex v_i to some vertex v_j with $j > i$. The cases of *downward* EA queries can be managed by a similar data structure rooted in v_0. LD queries and reachability queries can be recast as EA queries as already discussed in the preliminaries and in Lemma 1.

The naive solution to represent all possible upward earliest arrival paths in F is that of building a forest of rooted trees similar to the one shown on the top-left of Figure 1. This forest represents each label ℓ of each edge $e = (v_i, v_{i+1})$ in F as a node whose parent corresponds to the first label of (v_{i+1}, v_{i+2}) that is larger than or equal to ℓ (if any). Intuitively, moving upwards in F along an earliest arrival path corresponds to moving upwards in a tree of such

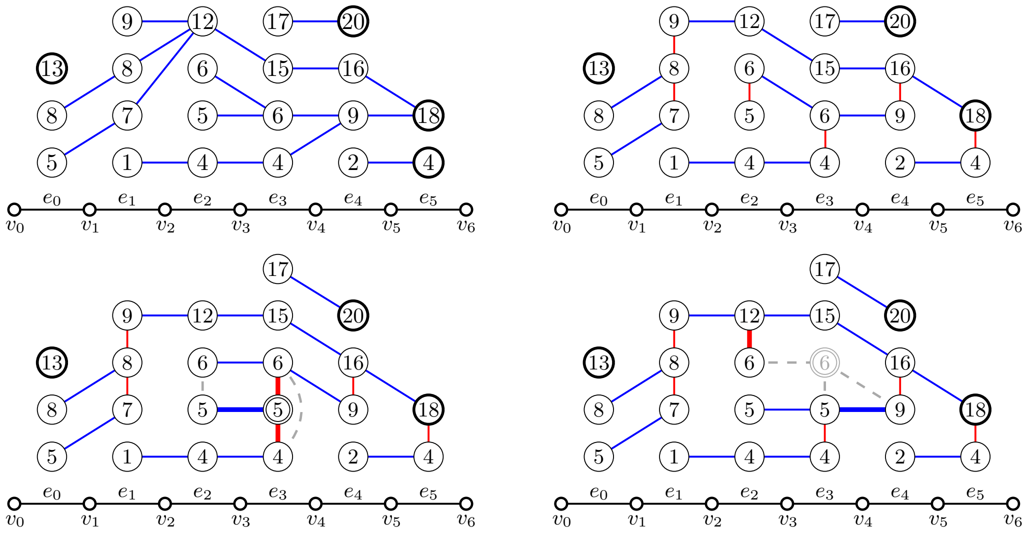


Figure 1 Top left: a forest representing all possible upward earliest arrival paths. Top right: the corresponding weighted forest $\mathcal{F}$. Red edges have weight 0, while blue ones have weight 1. Here, a label $\ell \in \lambda(e_i)$ is shown above edge e_i and models the node (ℓ, i). Bottom left and bottom right: the forests $\mathcal{F}$ obtained after adding label 5 to $\lambda(e_3)$ and then deleting label 6 from λe_3. New edges are shown in bold while removed ones are dashed.

a forest. Unfortunately, explicitly maintaining this forest would be costly since even a single label insertion/deletion could cause a large number of edges to be rewired. We circumvent this problem by maintaining a different edge-weighted forest $\mathcal{F}(\lambda)$ that still represents all possible upward earliest arrival paths while guaranteeing that each node has constant degree.

Let e_i be the edge (v_i, v_{i+1}). Our forest $\mathcal{F}(\lambda)$ contains a *node* (ℓ, i) for each edge e_i in F, and for each $\ell \in \lambda(e_i)$. To aid readability we use the term *vertex* to refer to vertices in F, and the term *node* for those in $\mathcal{F}(\lambda)$. To describe the edges of $\mathcal{F}(\lambda)$, we first define a partial function $\sigma_i(\ell, \lambda)$ that maps the labels $\ell \in \lambda(e_i)$ to a "successor" node, which is either the one corresponding to the smallest label strictly larger than ℓ on the same edge e_i, or the one corresponding to the successor of ℓ among the labels of edge e_{i+1}. More precisely, given $i \in \{0, \dots, n-2\}$ and $\ell \in \lambda(e_i)$, we consider $\ell^+ = \text{succ}(\ell + 1, \lambda(e_i))$. For $i = n-2$, $\sigma_i(\ell, \lambda)$ is defined as (e_i, ℓ^+) if $\ell^+ < +\infty$, and is undefined if $\ell^+ = +\infty$. For $i \in \{0, \dots, n-3\}$, we compare ℓ^+ with $\ell' = \text{succ}(\ell, \lambda(e_{i+1}))$. If $\ell^+ = \ell' = +\infty$, then $\sigma_i(\ell, \lambda)$ is undefined. Otherwise $\sigma_i(\ell, \lambda) = \begin{cases} (\ell^+, i) & \text{if } \ell^+ \le \ell'; \\ (\ell', i+1) & \text{if } \ell' < \ell^+. \end{cases}$

All the nodes (ℓ, i) for which $\sigma_i(\ell, \lambda)$ is undefined are the roots of their respective trees in $\mathcal{F}(\lambda)$. Whenever λ is clear from context we write $\mathcal{F}$ in place of $\mathcal{F}(\lambda)$ and $\sigma_i(\ell)$ in place of $\sigma_i(\ell, \lambda)$. See Figure 1 (top-right) for an example.

The weights of all edges of the form $((\cdot, i), (\cdot, i+1))$ are set to 1, while those of the remaining edges, of the form $((\cdot, i), (\cdot, i))$, are set to 0. Our definition of σ_i ensures that different children of the same node v in $\mathcal{F}$ must be linked to v using edges of different weights. As a consequence, each vertex has at most 2 children in $\mathcal{F}$.

Our data structure stores:

- A *top tree* [3] representing $\mathcal{F}$. A top-tree is a data structure capable of maintaining an edge-weighted forest under insertion of new nodes, deletion of singleton nodes, and under link and cut operations, i.e., addition and deletion of edges. Moreover, it supports weighted level ancestor queries: given a vertex v and $w \in \mathbb{N}$, it reports the w-th weighted level ancestor $\mathrm{LA}(v, w)$ of v in $\mathcal{F}$, i.e., the deepest ancestor of v at distance at least w from v (if any). Each of the above operations requires $O(\log \eta)$ worst-case time, where η is the number of nodes in the forest. A top tree with η nodes can be built in $O(\eta)$ time. Whenever a node/edge is added removed from $\mathcal{F}$, we perform the corresponding update operation on the backing top tree.

- A dictionary D_i for each edge e_i that supports insertions, deletions, and predecessor/successor queries in $O(\log \eta)$ worst-case time per operation, where η is the number of keys in D_i. The dictionary can be build in $O(\eta \log \eta)$ time. The keys in D_i are the labels in $\lambda(e_i)$ and each label ℓ is associated with a pointer to vertex (ℓ, i) in $\mathcal{F}$. Such a dictionary can be implemented using any dynamic balanced binary search tree.

Answering EA queries. To report $\mathrm{EA}(v_i, v_j, t)$ with $j > i$, we first find $\ell = \mathrm{succ}(t, \lambda(e_i))$ using D_i, then we query $\mathcal{F}$ for the $(j - i - 1)$-th weighted level ancestor (e_{j-1}, ℓ^*) of (e_i, ℓ). If such ancestor exist, we answer with ℓ^*. Otherwise we answer with $+\infty$. Finding ℓ requires time $O(\log L)$ in the worst case, while querying $\mathcal{F}$ can be done in worst-case $O(\log M)$ time. Hence the overall worst-case time required to answer the query is $O(\log M)$.

An auxiliary procedure. A label insertion causes the addition of a new node into $\mathcal{F}$, while a label deletion causes the deletion of the corresponding node from $\mathcal{F}$. This may result in some nodes in $\mathcal{F}$ that have an incorrect parent (or lack of thereof) from the one that would be expected according to the construction discussed above.

To address this problem, we define an auxiliary procedure $\mathrm{FixParent}(\ell, i)$ that will be helpful in restoring the correct state of $\mathcal{F}$. Such procedure takes the index i of some edge e_i in F and a label $\ell \in \lambda(e_i)$, and ensures that the edge from node (ℓ, i) to its parent in $\mathcal{F}$ (if any) is properly set (or unset) by only considering D_i and D_{i+1}.

To implement $\mathrm{FixParent}(\ell, i)$ we first *cut* the current edge from (ℓ, i) to its parent in $\mathcal{F}$ (if any). Then, we compute $\sigma_i(\ell)$ in $O(\log L)$ time by searching for the needed successors in D_i and D_{i+1}, according to the definition of σ_i. Finally, we *link* vertex (ℓ, i) with $\sigma_i(\ell)$, if any, in $\mathcal{F}$. Since link and cut operations also require time $O(\log M)$, the overall worst-case time spent by FixParent is $O(\log M)$.

Label addition. To add label ℓ on edge e_i, we first insert node (ℓ, e_i) into $\mathcal{F}$, and ℓ into D_i. Then, if $i \geq 1$ and $\ell' = \mathrm{pred}(\ell, \lambda(e_{i-1}))$ exists, we perform $\mathrm{FixParent}(\ell, i - 1)$. Next, we perform $\mathrm{FixParent}(\ell, i)$. Finally, if $\ell^- = \mathrm{pred}(\ell - 1, \lambda(e_i))$ exists, we perform $\mathrm{FixParent}(\ell^-, i)$. See Figure 1 for an example.

The overall worst-case time required is $O(\log M)$ since we only perform a constant number of predecessor/successor lookups, node insertions into $\mathcal{F}$, and calls to FixParent.

Label deletion. To remove label ℓ from edge e_i, we first delete ℓ from D_i and remove node (e_i, ℓ), along with all its incident edges, from $\mathcal{F}$. Then, if $i \geq 1$ and $\ell' = \mathrm{pred}(\ell, \lambda(e_{i-1}))$ exists, we perform $\mathrm{FixParent}(\ell', i - 1)$. Finally, if $\ell^- = \mathrm{pred}(\ell - 1, \lambda(e_i))$ exists, we perform $\mathrm{FixParent}(\ell^-, i)$. The overall worst-case time required is $O(\log M)$ since we only perform a constant number of predecessor/successor lookups, edge/node deletions from $\mathcal{F}$, and calls to FixParent.

4 Our data structure for temporal forests

In this section we discuss how our data structure for temporal paths with static topology can be generalized to temporal forests of rooted trees while also supporting link and cut operations as well as insertions and deletions of singleton vertices. More precisely, we prove the following result.

▶ **Theorem 3.** *Given a temporal forest of rooted trees, it is possible to build, in $O(n+M \log L)$ time, a data structure of linear size, where n is the number of vertices, L is the maximum number of labels on the same temporal edge, and M is the total number of (not necessarily distinct) labels in the forest at the time of the operation. The data structure supports label additions, label deletions, link, and cut operations, and addition/deletion of singleton vertices in $O(\log M)$ worst-case time per operation, EA and LD queries in $O(\log L \cdot \log M)$ worst-case time per operation, and reachability queries in $O(\log M)$ worst-case time per operation.*

We observe that it is easy to extend Theorem 2 to *fixed* temporal forests where one only needs to support label additions and deletions and aims for a query time of $O(\log n \cdot \log M)$, where n is the number of vertices in F. Indeed, it suffices to construct the data structure for temporal paths on each path of a heavy-light decomposition [21] of (each tree of) F. Details are given in the full version of the paper.

This section is organized as follows: we first discuss a data structure that only supports updates are label additions and deletions, and then we show how to handle link and cut operations as well as insertions and deletions of singleton vertices.

4.1 A data structure supporting only label additions and deletions

While the natural generalization of our construction for temporal paths of Section 3 to the case of trees would already provide a data structure supporting fast upward EA queries, such a solution runs into a similar problem as the one discussed for the naive approach. Indeed, in a tree of degree Δ a node of $\mathcal{F}$ may have $\Omega(\Delta)$ children, and its removal may cause the rewiring of $\Omega(\Delta)$ edges.

Our construction avoids this problem by further grouping the (nodes in $\mathcal{F}$ corresponding to the) edges from sibling vertices to their common parent v in F into a *block* $\mathcal{B}_v$ (see Figure 2).

For a non-root vertex $v \in V(F)$, we denote by $p(v)$ the parent of v in F, and by e_v the edge $(v, p(v))$. The forest $\mathcal{F}(\lambda)$ contains a *node* (ℓ, v) for each non-root vertex $v \in V(F)$ and for each $\ell \in \lambda(e_v)$. Moreover, for every non-leaf vertex $v \in V(F)$, we define the *block of* v as the set $\mathcal{B}_v(\lambda)$ containing all nodes (ℓ, u) where u is a child of v in F. We fix an arbitrary strict total ordering of the vertices of $V(F)$ and we think of the nodes of $\mathcal{F}$ as being ordered w.r.t. the order relation that compares nodes lexicographically, i.e., (ℓ, u) precedes (ℓ', v) if either $\ell < \ell'$, or if $\ell = \ell'$ and u precedes v in the chosen ordering of the vertices.

Similarly to the case of temporal paths, the edges of $\mathcal{F}(\lambda)$ are defined by making use of a partial function $\sigma_u(\ell, \lambda)$ that associates the nodes of $\mathcal{F}$ to their parents. More precisely, given a non-root vertex $u \in V(F)$ and $\ell \in \lambda(e_u)$, we let $v = p(u)$ so that $(\ell, u) \in \mathcal{B}_v$. If v is a root in F, $\sigma_u(\ell, \lambda)$ is defined as the strict successor of (ℓ, u) in $\mathcal{B}_v$, if it exists, otherwise $\sigma_u(\ell, \lambda)$ is undefined.

To define $\sigma_u(\ell, \lambda)$ when v is not a root of F, let $\ell' = \mathrm{succ}(\ell, \lambda(e_v))$. We distinguish two cases depending on whether the strict successor (ℓ^+, u^+) of (ℓ, u) in $\mathcal{B}_v$ exists. If (ℓ^+, u^+) exists we define $\sigma_u(\ell, \lambda) = \begin{cases} (\ell^+, u^+) & \text{if } \ell^+ \leq \ell'; \\ (\ell', v) & \text{if } \ell^+ > \ell'. \end{cases}$

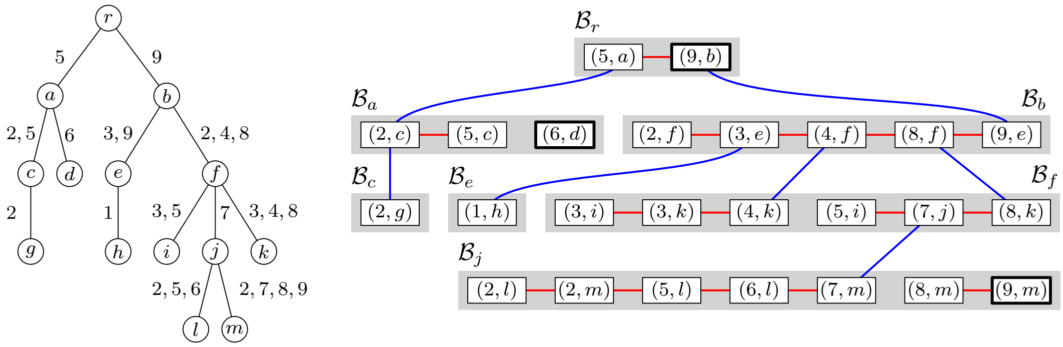


Figure 2 Left: a sample rooted temporal tree. Right: a representation of the forest $\mathcal{F}$ maintained by the data structure for temporal forests of Section 4.

If (ℓ^+, u^+) does not exist, then $\sigma_u(\ell, \lambda) = (\ell', v)$ when $\ell' < +\infty$ and $\sigma_u(\ell, \lambda)$ is undefined when $\ell' = +\infty$. The nodes (ℓ, u) such that $\sigma_u(\ell, \lambda)$ is undefined are the roots of the corresponding trees in $\mathcal{F}(\lambda)$. Whenever λ is clear from context, we write $\mathcal{F}$ in place of $\mathcal{F}(\lambda)$, $\mathcal{B}_v$ in place of $\mathcal{B}_v(\lambda)$, and $\sigma_u(\ell)$ in place of $\sigma_u(\ell, \lambda)$.

We say that an edge in $\mathcal{F}$ between nodes in the same block is *red*, while inter-block edges are *blue*. We assign weight 0 to red edges, and weight 1 to blue edges. See Figure 2 for an example.

Our data structure stores:

- A top tree representing $\mathcal{F}$. In addition to the operations discussed in Section 3, a top tree also supports lowest common ancestor (LCA) queries, i.e., given two nodes u, v of $\mathcal{F}$, it either reports that u and v belong to different trees in $\mathcal{F}$, or it answers with the deepest vertex w in the unique tree T containing both u and v such that w is an ancestor of both u and v. An LCA query requires $O(\log \eta)$ worst-case time, where η is the number of nodes in $\mathcal{F}$.
- A dictionary D_v for each non-root vertex $v \in V(F)$ that stores a key for each label in $\lambda(e_v)$ and that supports insertions, deletions and predecessor/successor queries. Each key ℓ stores, as satellite data, a pointer to node (ℓ, v) in $\mathcal{F}$.
- A dictionary for each block $\mathcal{B}_v$, where v is a non-leaf vertex in F. Such a dictionary stores all elements in $\mathcal{B}_v$ and supports insertions, deletions and predecessor/successor queries w.r.t. our order relation on the nodes.
- The depth $d(v)$ of each vertex v in the (unique) rooted tree T containing v in F, i.e., the hop distance between v and the root of T.

Adapting FixParent. FixParent(ℓ, v) takes a non-root vertex $v \in V(F)$ and a label $\ell \in \lambda(e_v)$, and ensures that the edge from node (ℓ, v) to its parent in $\mathcal{F}$, if any, is properly set.

To implement FixParent(ℓ, v) we first *cut* the current edge from (ℓ, v) to its parent in $\mathcal{F}$ (if any). Then we compute $\sigma_v(\ell)$ in $O(\log L)$ worst-case time by searching for the needed successors in $\mathcal{B}_{p(v)}$ and $D_{p(v)}$ (if $p(v)$ exists), according to the definition of σ_v. Notice that $\sigma_v(\ell)$ might be undefined. Finally, if $\sigma_v(\ell)$ is defined, we *link* vertex (ℓ, v) with $\sigma_v(\ell)$. Since link and cut operations require time $O(\log M)$, the overall worst-case time spent by FixParent is $O(\log M)$.

Label addition. To add label ℓ on edge e_v, we first insert node (ℓ, v) into $\mathcal{F}$, ℓ into D_v, and (ℓ, v) into $\mathcal{B}_{p(v)}$. Then, if v is not a leaf in F and $(\ell', u) = \text{pred}((\ell, +\infty), \mathcal{B}_v)$ exists, we perform FixParent(ℓ', u).[1] Next, we perform FixParent(ℓ, v). Finally, if the strict predecessor (ℓ^-, v^-) of (ℓ, v) in $\mathcal{B}_{p(v)}$ exists, we perform FixParent(ℓ^-, v^-).[2]

Label deletion. To delete label ℓ from edge e_v, we remove the node (ℓ, v) from $\mathcal{F}$ along with all its incident edges, we delete ℓ from D_v, and we delete (ℓ, v) from $\mathcal{B}_{p(v)}$. Then, if v is not a leaf in F and $(\ell', u) = \text{pred}((\ell, +\infty), \mathcal{B}_v)$ exists, we perform FixParent(ℓ', u). Finally, if the strict predecessor (ℓ^-, v^-) of (ℓ, v) in $\mathcal{B}_{p(v)}$ exists, we perform FixParent(ℓ^-, v^-).

To prove the correctness of our label addition deletion procedures we need the following lemma, whose proof is given in the full version of the paper, which captures the changes to $\mathcal{F}$ following an update:

▶ **Lemma 4.** *Consider a forest F, two functions $\lambda', \lambda : E(F) \to \mathscr{P}(\mathbb{Z})$ and an edge $e_v \in E(F)$ such that, for each $e' \in E(F) \setminus \{e_v\}$, $\lambda(e') = \lambda'(e')$ and $\lambda(e_v) = \lambda'(e_v) \setminus \{\ell\}$ with $\ell \in \lambda'(e_v)$. Let $\mathcal{F}' = \mathcal{F}(\lambda')$ and $\mathcal{F} = \mathcal{F}(\lambda)$. Let U be the set containing (ℓ, v), $(\ell', u) = pred((\ell, +\infty), \mathcal{B}_v(\lambda))$ (if it exists), and the strict predecessor (ℓ^-, v^-) of (ℓ, v) in $\mathcal{B}_{p(v)}(\lambda)$ (if it exists).*

We have that $V(\mathcal{F}) = V(\mathcal{F}') \setminus \{(\ell, v)\}$ and that all nodes in $V(\mathcal{F}) \setminus U$ have the same parent (or lack of thereof) in both F and $\mathcal{F}'$.

Then, the correctness of the label addition procedure follows from Lemma 4 with λ (resp. λ') chosen as the function that maps each edge to its labels before (resp. after) the addition, once one observe that FixParent is invoked on all vertices of the set U defined by the lemma. Symmetrically, the correctness of the label deletion procedure follows from Lemma 4 when the roles of λ and λ' are reversed.

The overall worst-case time required per update is $O(\log M)$ since we only perform a constant number of (strict) predecessor lookups, edge/node deletions from $\mathcal{F}$, and calls to FixParent. Observe that Lemma 4 implies that, when node (ℓ, v) is deleted, its degree in $\mathcal{F}$ is constant (since each child of (ℓ, v) changes its parent following the deletion).

Answering upward EA queries and downward LD queries. We first discuss how to report EA(u, v, t) when u is a proper descendent of v in F.[3] To do so, we start by finding $\ell = \text{succ}(t, \lambda(e_u))$. If $\ell = +\infty$, we answer with $+\infty$, otherwise we query $\mathcal{F}$ for the $(d_v - d_u - 1)$-th weighted level ancestor of (ℓ, u). If such an ancestor (ℓ^*, w) exists we answer with ℓ^*, otherwise we answer with $+\infty$. This requires $O(\log M)$ worst-case time since it only involves a successor lookup in D_u and a weighted level ancestor query on the top tree representing $\mathcal{F}$.

The correctness of our query procedure stems by a structural property of $\mathcal{F}$ captured by the following lemma, whose proof is given in the full version of the paper.

▶ **Lemma 5.** *Let $u, v \in V(F)$ where v is a proper ancestor of v, let $t \in \mathbb{Z} \cup \{-\infty\}$, and define $\ell = succ(t, \lambda(e_u))$. If (i) $\ell = +\infty$ or (ii) $\ell < +\infty$ and $LA((\ell, u), d_v - d_u - 1)$ does not exist, then $EA(u, v, t) = +\infty$. Otherwise, $LA((\ell, u), d_v - d_u - 1) = (\ell^*, w)$ where w is the unique ancestor of u such that $p(w) = v$ and $\ell^* = EA(u, v, t)$.*

[1] Notice that the set of nodes in B_v does not change following the label addition.

[2] Notice that the insertion of (ℓ, v) into $B_{p(v)}$ does not affect the value (nor the existence) of the strict predecessor of (ℓ, v) in $B_{p(v)}$.

[3] Since the topology of F does not change, checking whether u is a proper descendent of v can be done in constant time after $O(n)$-time preprocessing, where n is the number of vertices in F.

To answer downward LD queries, we maintain a mirrored data structure for $-F$, in which each original label ℓ is replaced with $-\ell$. Lemma 1 allows us to answer downward LD queries in $O(\log M)$ worst-case time by performing upward EA queries on the data structure for $-F$.

Answering upward LD queries and downward EA queries. To report $\mathrm{LD}(u, v, t)$ when u is a proper descendent of v in F, we binary search for the largest label $\ell^* \in \lambda(e_u)$ such that $\mathrm{EA}(u, v, \ell^*) \leq t$. Then, we report ℓ^*. The correctness of our query immediately follows from the fact that the values $\mathrm{EA}(u, v, \ell)$ are monotonically non decreasing w.r.t. ℓ. This requires a worst-case time of $O(\log L \cdot \log M)$. To report $\mathrm{EA}(w, v, t)$ when v is a proper descendant of w in F, we compute $\mathrm{LD}(v, w, -t)$ on the data structure for $-F$ and answer with $-\mathrm{LD}(v, w, -t)$. The worst-case time required is $O(\log L \cdot \log M)$.

Answering general EA and LD queries. To answer a general $\mathrm{EA}(u, v, t)$ query we compute the lowest common ancestor w of u and v in F in constant time using the data structure in [5], and we return $\mathrm{EA}(w, v, \mathrm{EA}(u, v, t))$ where $\mathrm{EA}(w, v, \cdot)$ is a downward query, and $\mathrm{EA}(u, v, \cdot)$ is an upward query. General LD queries can similarly be answered by using the data structure for $-F$. The worst-case time required to answer such queries is $O(\log L \cdot \log M)$.

Answering temporal reachability queries. To report whether a vertex v is reachable from a vertex u using a temporal path in F that departs no earlier than t_d and arrives no later t_a, we compute the lowest common ancestors w of u and v in F in constant time and we answer affirmatively iff $\mathrm{EA}(u, w, t_d) \leq \mathrm{LD}(w, v, t_a)$. The overall worst-case time required is $O(\log M)$ since we only need to perform an upward EA query and a downward LD query.

4.2 Supporting link and cut operations

To support general *link* and *cut* operations we additionally maintain F using a top tree.[4] Each time that an edge is added/removed from F, we perform the corresponding operation on the backing top tree. Moreover, we no longer explicitly maintain the depths d_v, but rather we query the top tree of F every time any such depth is needed.[5] Similarly, all LCA queries on F are now performed using the backing top tree.

Insertions and deletion of singleton vertices are straightforward, therefore we only discuss how to handle link and cut operations.

Link operations. To implement a link operation where u is the root of some tree T in F and v is some vertex of a tree T' of F with $T' \neq T$, and a label $\ell \in \mathbb{Z}$, we first *link* u and v in F by adding edge (u, v), so that the parent of u becomes v, and we add label ℓ to (u, v) as explained in Section 4.1. The worst-case time required is $O(\log M)$.

Cut operations. To cut an edge (u, v) of F with a single label $\ell \in \lambda((u, v))$, where u is a child of v w.l.o.g., we first remove the only label ℓ (corresponding to the only key in D_u) from (u, v) as explained in Section 4.1. Then, we *cut* (u, v) from F, thus creating a new tree rooted in u. The worst-case time required is $O(\log M)$.

[4] Some technical care is needed to obtain the stated bounds, which only depend on M, when $M = o(n)$. This can be achieved by not actually storing singleton vertices in the top tree, so that the number of vertices in the top tree is always $O(M)$. The operations of our data structure are easy to adapt to handle this edge case. E.g., whenever a link operation on F involves some singleton vertex v, we can add v to the top tree immediately before performing the link.

[5] Indeed, the top tree can report the root of the tree in F that contains v and the hop-distance between any two nodes in F in logarithmic time.

5 Our data structure for temporal forests with latencies

Our model of temporal graphs can be generalized by introducing *latencies*. In *temporal graphs with latencies*, each edge e is associated with a collection of pairs (ℓ, d) encoding that edge e can be traversed at the *departure time* ℓ stating from one of its endvertices in order to reach the other endvertex at time $\ell + d$. The value d is called a *latency*.

Equivalently, each (activation time, latency) pair (ℓ, d) can be expressed as (ℓ, α), where $\alpha = \ell + d$ is the *arrival time*. These two representations are clearly equivalent, but the latter one results in a lighter notation for our purposes. Therefore, in the rest of the section we will adopt the (departure time, arrival time) convention and we accordingly define $\lambda(e)$ as the set of all (departure time, arrival time) pairs (ℓ, α) associated with edge e.[6]

A *temporal path* π from vertex $u \in V(F)$ to vertex $v \in V(F) \setminus \{u\}$ in F is a sequence of triples $\langle (e_1, \ell_1, \alpha_1), (e_2, \ell_2, \alpha_2), \dots, (e_k, \ell_k, \alpha_k) \rangle$ such that $\langle e_1, e_2, \dots, e_k \rangle$ is a path from u to v in F, $(\ell_i, \alpha_i) \in \lambda(e_i)$ for all $i = 1, \dots, k$, and $\alpha_i \le \ell_{i+1}$ for all $i = 1, \dots, k-1$. The departure time of π is ℓ_1 and its arrival time is α_k. The notions of earliest arrival paths, latest departure paths, reachability, and the corresponding queries extend naturally.

In this section we argue that our data structure for temporal forests can be adapted to additionally support latencies. This comes at the cost of turning our worst-case bounds on the time complexities of the link and cut operations into amortized bounds that hold for the incremental case, in which only link operations are allowed, and in the decremental case, in which only cut operations are allowed.[7]

▶ **Theorem 6.** *Given a temporal forest of rooted trees with latencies, it is possible to build a data structure of linear size that supports EA and LD queries in $O(\log L \cdot \log M)$ worst-case time per operation, and reachability queries in $O(\log M)$ worst-case time per operation, where L is the maximum number of labels on the same temporal edge, and M is the total number of (not necessarily distinct) labels in the forest at the time of the operation. In the incremental case, the data structure also supports insertions of singleton vertices, label insertions, and link operations in amortized time $O(\log M)$. In the decremental case, the data structure also supports deletions of singleton vertices, label deletions, and cut operations in amortized $O(\log M)$ time and amortized building time of $O(M \log L)$.*[8]

As before, we consider a temporal forest with latencies F containing rooted temporal trees and, for a non-root vertex v, we define e_v as the edge from v to its parent $p(v)$ in the unique tree of F containing v. We define $\mathcal{F}(\lambda)$ as the forest containing a *node* (α, ℓ, v) for each non-root vertex v and for each $(\ell, \alpha) \in \lambda(e_v)$. For each non-leaf vertex $v \in V(F)$, we also define the *block of* v as the set $\mathcal{B}_v(\lambda)$ containing all nodes (α, ℓ, u) such that $p(u) = v$. We fix an arbitrary order of the vertices of $V(F)$ and we think of the nodes of $\mathcal{F}$ as being ordered w.r.t. the order relation that compares nodes lexicographically, i.e., (α, ℓ, u) precedes (α', ℓ', v) if (i) $\alpha < \alpha'$, or (ii) $\alpha = \alpha'$ and $\ell < \ell'$, or (iii) $\alpha = \alpha'$, $\ell = \ell'$, and u precedes v in the chosen ordering of the vertices. Informally, we first order the nodes in a block by arrival time, then by departure time, and finally by their corresponding vertex in F.

[6] The special case in which all labels (ℓ, α) have $\alpha = \ell$ corresponds to the model used in the previous sections.

[7] Our data structure can still handle arbitrary sequences of link and cut operations, but the amortized bounds do not hold for this case.

[8] In the decremental case, we can naturally assume that the initial number of vertices n in F satisfies $n = \Omega(M)$.

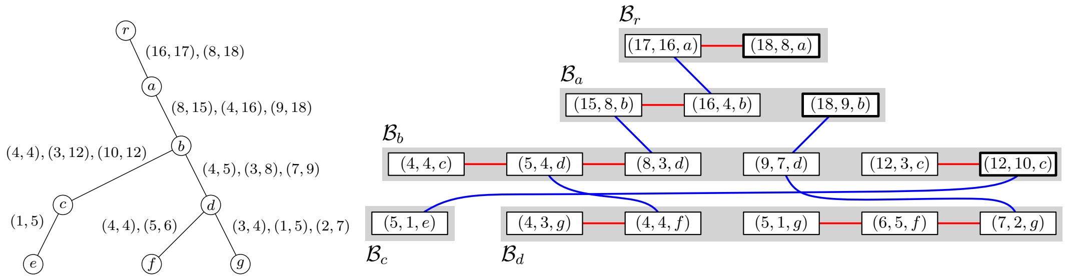


Figure 3 Left: a sample rooted temporal tree with latencies. Right: a representation of the forest $\mathcal{F}$ maintained by the data structure for temporal forests with latencies of Section 5.

We now define the analogue of the function σ for temporal forests with latencies. More precisely, given a non root vertex u in F with parent $v = p(u)$ and $(\ell, \alpha) \in \lambda(e_u)$, we let (α^+, ℓ^+, u^+) be the *strict successor* of (α, ℓ, u) in $\mathcal{B}_v(\lambda)$, if any. If v is a root of a tree in F, then $\sigma_u(\alpha, \ell, \lambda) = (t^+, \ell^+, u^+)$ if (t^+, ℓ^+, u^+) exists, otherwise $\sigma_u(\alpha, \ell, \lambda)$ is undefined. When v is not a root of any tree in F, we define $\text{NextHop}(\alpha, \ell, u)$ as the node (α', ℓ', v) such that $(\ell', \alpha') \in \lambda(e_v)$, $\ell' \geq \alpha$, and α' is minimized, breaking ties in favor of labels with the largest departure time (such a node might not exists).

We observe that if $\text{NextHop}(\alpha^+, \ell^+, u^+)$ exists, if it exists, either coincides with (α', ℓ', v) or it follows (α', ℓ', v) in $\mathcal{B}_{p(v)}$ w.r.t. our ordering. Thus, if both (α^+, ℓ^+, u^+) and (α', ℓ', v) exist, we define: $\sigma_u(\alpha, \ell, \lambda) = \begin{cases} (\alpha^+, \ell^+, u^+) & \text{if } \alpha^+ \leq \ell'; \\ (\alpha', \ell', v) & \text{if } \ell' < \alpha^+. \end{cases}$
Here the condition $\alpha^+ \leq \ell'$ is equivalent to the following: $\text{NextHop}(\alpha^+, \ell^+, u^+)$ exists and coincides with (α', ℓ', v). If neither (α^+, ℓ^+, u^+) nor (α', ℓ', v) exist, then $\sigma_u(\alpha, \ell, \lambda)$ is undefined and thus (α, ℓ, λ) is a root. Otherwise, $\sigma_u(\alpha, \ell, \lambda)$ is defined as the only node that exists among (α^+, ℓ^+, u^+) and (α', ℓ', v). As usual, we drop the parameter λ from $\mathcal{F}(\lambda)$, $\sigma_u(\alpha, \ell, \lambda)$, and $\mathcal{B}(\lambda)$ whenever λ is clear from context.

Our data structure is analogous to that of Section 4, with the following exceptions:

- each dictionary D_v storing the pairs in $\lambda(e_v)$ now supports queries of the following form: given a range of values of interest for ℓ (resp. α), return the minimum/maximum value of α (resp. ℓ) w.r.t. all labels $(\ell, \alpha) \in \lambda(e_v)$ such that ℓ (resp. α) is in the sought range (notice that value might not exist). This can be done in time $O(\log |\lambda(e_v)|)$ using, e.g., priority search trees [20], which require space $O(|\lambda(e_v)|)$.
- for each non-leaf node v we say that a node $(\alpha, \ell, u) \in \mathcal{B}_v$ is a *head* of $\mathcal{B}_v$ if (α, ℓ, u) either has no parent in $\mathcal{F}$ or it is linked to its parent with a blue edge. We store a dictionary $\mathcal{H}_v$ that contains all heads of $\mathcal{B}_v$.

Since queries are analogous to the latency-free case, we only focus on label additions/deletions.

An auxiliary procedure. The auxiliary procedure $\text{FixParent}(\ell, \alpha, v)$ for the case with latencies is similar to FixParent in the case without latencies, since it only uses the definition of σ_v as before. The only difference is that that the execution of FixParent also needs to update $\mathcal{H}_{p(v)}$ taking into account the new parent of (ℓ, α, v).

Label addition. To add label (ℓ, α) on edge e_v, we first insert node (α, ℓ, v) into $\mathcal{F}$, (ℓ, α) into D_v, and (α, ℓ, v) into $\mathcal{B}_{p(v)}$. We find the maximum value ℓ^- of ℓ'' among all pairs $(\ell'', \alpha'') \in \lambda(e_v)$ with $\alpha'' < \alpha$ using D_v.

Let $(\alpha_1, \ell_1, u_1), \ldots, (\alpha_k, \ell_k, u_k)$ be all the nodes in $\mathcal{B}_v$ such that $\ell^- < \alpha_i \leq \ell$ sorted w.r.t. our order (see Figure 4). We observe that all these nodes are consecutive in $\mathcal{B}_v$. The parent of (α_k, ℓ_k, u_k) becomes (α, ℓ, v), while the parent of all (α_i, ℓ_i, u_i) becomes $(\alpha_{i+1}, \ell_{i+1}, u_{i+1})$. This requires updating all heads in $\mathcal{H}_v$ between (α_1, ℓ_1, u_1) and $(\alpha_{k-1}, \ell_{k-1}, u_{k-1})$, and can be in time $O(h \log M)$, where h is the number of such heads. This causes all the updated heads to be removed from $\mathcal{H}_v$, and (α_k, ℓ_k, u_k) to become a new head (if that was not already the case). Next, we perform FixParent(α, ℓ, v). Finally, if the strict predecessor (α^*, ℓ^*, v^*) of (α, ℓ, v) in $\mathcal{B}_{p(v)}$ exists, we perform FixParent(α^*, ℓ^*, v^*).

We now argue that the amortized time complexity of each label insertion is $O(\log M)$ in the incremental case using the accounting method. Let $c > 0$ be a sufficiently large constant. We keep a coin of value at least $c \cdot H_M$, where $H_i = \sum_{j=1}^{i} \frac{1}{j}$ denotes the i-th harmonic number, on each node of $\mathcal{F}$ that is either a root or is linked to its parent with a blue edge. When a new label is inserted, we pay up to $cM \cdot \frac{1}{M+1}$ for the increase in value of the existing coins so that each coin has value cH_{M+1}, and cH_{M+1} for the coin on the new node (α, ℓ, v) in $\mathcal{F}$. We also add a coin of value cH_{M+1} on each of (α, ℓ, v) and (α_k, ℓ_k, u_k). Notice that each node of $(\alpha_1, \ell_1, u_1), \ldots, (\alpha_{k-1}, \ell_{k-1}, u_{k-1})$ that changes the edge towards its parent either had no parent, or it was linked to its previous parent with a blue edge. Moreover, the new edges towards its parent must be red. This means that such a node had a coin of value cH_{M+1} that we can use to pay for the cost of the rewiring. All the other operations performed during a label addition cost $O(\log M)$ worst-case time.

Label deletion. To delete the label (ℓ, α) from edge e_v, we remove the node (α, ℓ, v) from $\mathcal{F}$ along with all its incident edges, we delete (ℓ, α) from D_v, and we delete (α, ℓ, v) from $\mathcal{B}_{p(v)}$, and possibly from $\mathcal{H}_{p(v)}$.

If v is not a leaf in F, let $(\alpha_1, \ell_1, u_1), \ldots, (\alpha_k, \ell_k, u_k)$ be the nodes, in order, that had the NextHop equal to (α, ℓ, v). Notice that such nodes induce a red path in $\mathcal{F}$, and that (α_k, ℓ_k, u_k) was the unique blue child of (α, ℓ, v) before the deletion.

As a consequence of the deletion, some of the nodes in $(\alpha_1, \ell_1, u_1), \ldots, (\alpha_k, \ell_k, u_k)$ will become heads of $\mathcal{B}_v$, and hence will have a blue edge towards their new parent in $\mathcal{B}_{p(v)}$. We find (α_k, ℓ_k, u_k) in $O(\log M)$ worst-case time via binary search in $\mathcal{B}_v$ and we use it discover such heads as follows: Initially $(\alpha^*, \ell^*, u^*) = (\alpha_k, \ell_k, u_k)$, and $z = \text{NextHop}(\alpha_k, \ell_k, u_k)$. We binary search $\mathcal{B}_v$ for the rightmost node (α', ℓ', u') (w.r.t. our ordering) that precedes (α_k, ℓ_k, u_k) and is such that NextHop$(\alpha', \ell', u') \neq z$. We mark node (α', ℓ', u') as a head, and repeat the above procedure using $(\alpha^*, \ell^*, u^*) = (\alpha', \ell', u')$, and $z = \text{NextHop}(\alpha', \ell', u')$. We stop the above procedure as soon as z precedes (α, ℓ, v) in $B_{p(v)}$. Then this requires $O(\log M)$ time, plus and additional $O(\log M)$ time per discovered head since we can check whether NextHop$(\alpha', \ell', u') \neq z$ in constant time. Indeed, calling $z = (\alpha_z, \ell_z, v)$, we can define ℓ_z^- as the largest value of ℓ'' among the pairs $(\ell'', \alpha'') \in \lambda(e_v)$ with $\alpha'' < \alpha_z$ (such ℓ_z^- can be found by querying D_v). Then NextHop$(\alpha', \ell', u') \neq z$ iff $\ell' < \ell_z^-$.

We run FixParent on (α_k, ℓ_k, u_k) and on all the marked heads, which also updates $\mathcal{H}_v$.

Finally, if the strict predecessor (α^-, ℓ^-, v^-) of (α, ℓ, v) in $\mathcal{B}_{p(v)}$ exists, we perform FixParent(α^-, ℓ^-, v^-).

We now argue that the amortized time complexity of each label deletion is $O(\log M)$ in the decremental case using the accounting method. Let c be sufficiently large constant. We keep a coin of value at least $c \cdot H_M$ on each node of $\mathcal{F}$ that is either a root or is linked to its parent with a red edge. Hence, we pay an amortized cost of $O(M \log M)$ at construction time. When a label is deleted, we pay $2cH_M$ to add a coin on each of (α^-, ℓ^-, v^-), and

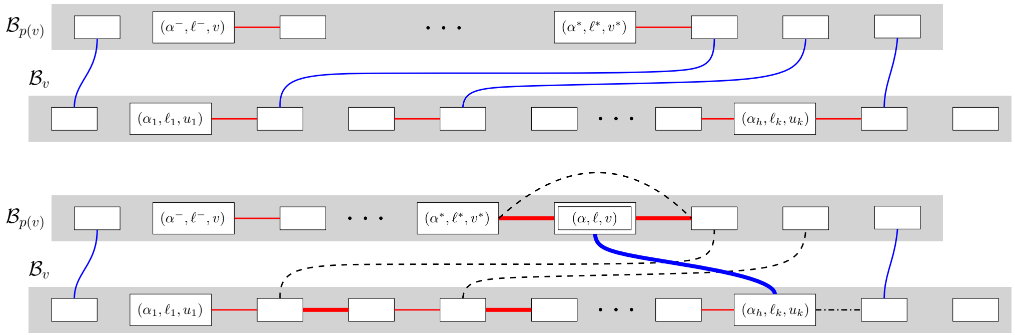


Figure 4 A qualitative representation of the changes resulting from the addition of label (ℓ,α) on edge e_v. On the top: the blocks $\mathcal{B}_{p(v)}$ and $\mathcal{B}_v$ before inserting node (α,ℓ,v). On the bottom: the new state of $\mathcal{B}_{p(v)}$ and $\mathcal{B}_v$. Bold lines represent new edges, while dashed lines represent removed ones.

(α_k,ℓ_k,u_k). Moreover, since each marked head (α',ℓ',u') had a red edge towards its parent before the deletion, we spend such a coin to pay for the execution of FixParent on (α',ℓ',u'). All the other operations performed during a label deletion cost $O(\log M)$ worst-case time.

Uniform latencies. A special case of temporal graphs with latencies is the one with *uniform latencies*, where all time labels have the same latency. In this case, the ordering defined on the nodes of $\mathcal{F}$ is exactly the same as the one used in Section 4, for the case without latencies. For this reason, the data structure we presented in this section guarantees a worst-case update time of $O(\log M)$ because only a constant number of nodes in $\mathcal{F}$ can change their parents.

▶ **Corollary 7.** *Given a temporal forest of rooted trees with uniform latencies, it is possible to build in $O(n + M\log L)$ time, a data structure of linear size, n is the number of vertices, L is the maximum number of labels on the same temporal edge, and M is the total number of (not necessarily distinct) labels in the forest at the time of the operation. The data structure supports label additions/deletions, link/cut operations and addition/deletion of singleton vertices in $O(\log M)$ worst-case time per operation, EA and LD queries in $O(\log L \cdot \log M)$ worst-case time per operation, and reachability queries in $O(\log M)$ worst-case time per operation.*

References

1 Eleni C. Akrida, George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, Paul G. Spirakis, and Viktor Zamaraev. How fast can we reach a target vertex in stochastic temporal graphs? *J. Comput. Syst. Sci.*, 114:65–83, 2020. doi:10.1016/J.JCSS.2020.05.005.

2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123, 2020. doi:10.1016/J.JCSS.2019.08.002.

3 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

4 Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Yessin M. Neggaz. Testing temporal connectivity in sparse dynamic graphs. *CoRR*, abs/1404.7634, 2014. arXiv:1404.7634.

5 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

6 Davide Bilò, Sarel Cohen, Tobias Friedrich, Hans Gawendowicz, Nicolas Klodt, Pascal Lenzner, and George Skretas. Temporal network creation games. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 2511–2519. ijcai.org, 2023. doi:10.24963/IJCAI.2023/279.

7 Davide Bilò, Gianlorenzo D'Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi. Sparse temporal spanners with low stretch. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.19.

8 Davide Bilò, Gianlorenzo D'Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi. Blackout-tolerant temporal spanners. *J. Comput. Syst. Sci.*, 141:103495, 2024. doi:10.1016/J.JCSS.2023.103495.

9 Luiz F. Afra Brito, Marcelo Keese Albertini, Arnaud Casteigts, and Bruno Augusto Nassif Travençolo. A dynamic data structure for temporal reachability with unsorted contact insertions. *Social Network Analysis and Mining*, 12, 2021. URL: https://api.semanticscholar.org/CorpusID:231847148.

10 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. doi:10.1007/S00453-021-00831-W.

11 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *J. Comput. Syst. Sci.*, 121:1–17, 2021. doi:10.1016/J.JCSS.2021.04.004.

12 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp thresholds in random simple temporal graphs. *SIAM J. Comput.*, 53(2):346–388, 2024. doi:10.1137/22M1511916.

13 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.141.

14 Thomas Erlebach and Jakob T. Spooner. Parameterised temporal exploration problems. *J. Comput. Syst. Sci.*, 135:73–88, 2023. doi:10.1016/J.JCSS.2023.01.003.

15 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 10193–10201. AAAI Press, 2022. doi:10.1609/AAAI.V36I9.21259.

16 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, 2022. doi:10.1145/3555806.

17 Petter Holme. Temporal networks. In *Encyclopedia of Social Network Analysis and Mining*, pages 2119–2129. Springer, 2014. doi:10.1007/978-1-4614-6170-8_42.

18 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in o(log n(loglog n)^2) amortized expected time. *TheoretiCS*, 2, 2023. doi:10.46298/THEORETICS.23.6.

19 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.

20 Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985. doi:10.1137/0214021.

21 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

22 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014. doi:10.14778/2732939.2732945.

Partitioning Problems with Splittings and Interval Targets

Samuel Bismuth
Department of Computer Science, Ariel University, Israel

Vladislav Makarov
Department of Mathematics and Computer Science, St. Petersburg State University, Russia

Erel Segal-Halevi
Department of Computer Science, Ariel University, Israel

Dana Shapira
Department of Computer Science, Ariel University, Israel

Abstract

The n-way number partitioning problem is a classic problem in combinatorial optimization, with applications to diverse settings such as fair allocation and machine scheduling. All these problems are NP-hard, but various approximation algorithms are known. We consider three closely related kinds of approximations.

The first two variants optimize the partition such that: in the first variant some fixed number s of items can be *split* between two or more bins and in the second variant we allow at most a fixed number t of *splittings*. The third variant is a decision problem: the largest bin sum must be within a pre-specified interval, parameterized by a fixed rational number u times the largest item size.

When the number of bins n is unbounded, we show that every variant is strongly NP-complete. When the number of bins n is fixed, the running time depends on the fixed parameters s, t, u. For each variant, we give a complete picture of its running time.

For $n = 2$, the running time is easy to identify. Our main results consider any fixed integer $n \geq 3$. Using a two-way polynomial-time reduction between the first and the third variant, we show that n-way number-partitioning with s split items can be solved in polynomial time if $s \geq n - 2$, and it is NP-complete otherwise. Also, n-way number-partitioning with t splittings can be solved in polynomial time if $t \geq n - 1$, and it is NP-complete otherwise. Finally, we show that the third variant can be solved in polynomial time if $u \geq (n - 2)/n$, and it is NP-complete otherwise. Our positive results for the optimization problems consider both min-max and max-min versions.

Using the same reduction, we provide a fully polynomial-time approximation scheme for the case where the number of split items is lower than $n - 2$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Number Partitioning, Fair Division, Identical Machine Scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.12

Related Version *Full Version*: https://arxiv.org/pdf/2204.11753 [4]

Funding *Samuel Bismuth*: Israel Science Foundation grant no. 712/20.
Erel Segal-Halevi: Israel Science Foundation grant no. 712/20.

Acknowledgements The paper started from discussions in the stack exchange network:
(1) https://cstheory.stackexchange.com/q/42275;
(2) https://cs.stackexchange.com/a/141322.
Dec-SplitItem$[3, 1](\mathcal{X})$ was first solved by Mikhail Rudoy using case analysis. The relation to FPTAS was raised by Chao Xu. We are also grateful to John L. https://cs.stackexchange.com/a/149567.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the classic setting of the n-way number partitioning problem, the inputs are a list $\mathcal{X} = (x_1, \ldots, x_m)$ of m non-negative integers and a number of bins n, and the required output is an n-way partition (a partition of the integers into n bins) that attains some pre-determined objective. In the decision version of the problem, the objective is to decide whether there exists an n-way partition of $\mathcal{X}$ such that every bin sum is exactly equal to $\sum_{x_i \in \mathcal{X}} x_i/n$ (we call it a *perfect partition*). In the min-max optimization version, the objective is to find an n-way partition of $\mathcal{X}$ such that the maximum bin sum is minimized, while in the max-min optimization version, the goal is to maximize the smallest bin sum.

For each problem of this paper, the problem objective is mentioned first, the fixed parameters in square brackets, and the problem input in parenthesis. Let us formally define the min-max version of the n-way number partitioning problem, where n is a fixed parameter:

MinMax-$\textsc{Part}[n](\mathcal{X})$*: Minimize* $\max(b_1, \ldots, b_n)$*, where* $b_1, \ldots, b_n$ *are sums of bins in an* n*-way partition of* $\mathcal{X}$*.*

When n is unbounded, Dec-$\textsc{Part}(n, \mathcal{X})$ (the decision version of the n-way number partitioning problem) is known to be strongly NP-hard (it is equivalent to 3-partition) [8]. And for every fixed $n \geq 2$ Dec-$\textsc{Part}[n](\mathcal{X})$ is known to be NP-hard [9]. In addition, many instances of the decision version are negative (there is no perfect partition). The latter reasons give us a good motivation to investigate variants of the n-way number partitioning problem, for which the running time complexity is better, and the number of positive instances (admitting a perfect partition) will significantly grow. We present three variants, that relax the initial problem to solve our concerns. The first two variants allow "divisible" items, bounded by some natural numbers s and t. We define the decision and the min-max versions as follows:

Dec-$\textsc{SplitItem}[n, s](\mathcal{X})$*: Decide if there exists a partition of* $\mathcal{X}$ *among* n *bins with at most* s *split items, such that* $\max(b_1, \ldots, b_n) \leq S$*.*

MinMax-$\textsc{SplitItem}[n, s](\mathcal{X})$*: Minimize* $\max(b_1, \ldots, b_n)$*, where* $b_1, \ldots, b_n$ *are sums of bins in an* n*-way partition of* $\mathcal{X}$ *in which at most* s *items are split.*

Dec-$\textsc{Splitting}[n, t](\mathcal{X})$*: Decide if there exists a partition of* $\mathcal{X}$ *among* n *bins with at most* t *splittings, such that* $\max(b_1, \ldots, b_n) \leq S$*.*

MinMax-$\textsc{Splitting}[n, t](\mathcal{X})$*: Minimize* $\max(b_1, \ldots, b_n)$*, where* $b_1, \ldots, b_n$ *are sums of bins in an* n*-way partition of* $\mathcal{X}$ *in which at most* t *splittings are allowed.*

The number of splittings is at least the number of split items but might be larger. For example, a single item split into 10 different bins counts as 9 splittings. Note that the problem definitions do not determine in advance which items will be split, but only bound their number, or bound the number of splittings. The solver may decide which items to split after receiving the input.

Our motivating application for the variants comes from fair division and machine scheduling. For fair division, some m items with market values $x_1, \ldots, x_m$ have to be divided among n agents. A *perfect partition* is one in which each partner receives a total value of exactly $\sum_i x_i/n$. When the original instance does not admit a perfect partition, we may want to *split* one or more items among two or more agents. Or, we can allow some *splittings.* Divisible items are widespread in fair division applications – the ownership of one or more items may be split to attain a perfectly fair partition. However, divisible items may be inconvenient or expensive. Therefore, the number of split items or splittings should be bounded. The same is true for machine scheduling, in which agents are considered as machines, and items as jobs. It is possible for a job to be divided and be processed by two machines simultaneously. The following examples tackle real-life fair division or machine scheduling problems.

(1) Consider $n = 2$ heirs who inherited $m = 3$ houses and have to divide them fairly. The house values are $\mathcal{X} = (100, 200, 400)$. If all houses are considered discrete, then an equal division is not possible. If all houses can be split, then an equal division is easy to attain by giving each heir 50% of every house, but it is inconvenient since it requires all houses to be jointly managed. A solution often used in practice is to decide in advance that *a single* house can be split (or only one splitting is allowed). In this case, after receiving the input, we can determine that splitting the house with a value of 400 lets us attain a division in which each heir receives the same value of 350. [1]

(2) Consider a food factory with $n = 3$ identical chopping machines, who has to cut $m = 4$ vegetables with processing times $\mathcal{X} = (10, 7, 5, 5)$ minutes. Each job is divisible as one vegetable may be cut in different machines, but splitting a job is inconvenient since it requires washing more dishes. Without splitting, the minimum total processing time is 10 minutes: $(10), (7), (5, 5)$. By splitting the vegetable with processing time 10 into three different machines, the processing time is 9 minutes: $(7, 2), (5, 4), (5, 4)$.

The third variant only admits a decision version, parameterized by a rational number $u \geq 0$:

*Dec-*Inter$[n, u](\mathcal{X})$*: Decide if there exists a partition of* $\mathcal{X}$ *into* n *bins with sums* $b_1, \ldots, b_n$ *such that* $S \leq \max(b_1, \ldots, b_n) \leq S + u \cdot M$*, where* $S := (\sum_i x_i)/n$ *and* $M := (\max_i x_i)/n$*.*

We will also use another definition of this variant, parameterized by a rational number $v \geq 0$:

*Dec-*Inter$[n, v](\mathcal{X})$*: Decide if there exists a partition of* $\mathcal{X}$ *into* n *bins with sums* $b_1, \ldots, b_n$ *such that* $S \leq \max(b_1, \ldots, b_n) \leq (1 + v) \cdot S$*, where* $S := (\sum_i x_i)/n$*.*

Note that when $u = vS/M$, both definitions are the same. In general, the runtime complexity of this problem depends on the size of the allowed interval (i.e., the interval $[S, S + u \cdot M]$): the problem is NP-complete when the interval is "small" and in P when the interval is "large". Specifically, when $n = 2$, the runtime complexity depends on the ratio of the allowed interval to the *bin sum*, while when $n \geq 3$ it depends on the ratio of the allowed interval to the *largest item*. We notice that, if we can solve Inter for any interval length in polynomial-time, then by binary search we can solve Part in polynomial-time; which is not possible unless P=NP. So in Inter, we look for the smallest interval for which we can decide in polynomial-time whether it contains a solution.

As an application example, consider the fair allocation of indivisible items among two agents. Suppose there is a small amount of money, that can be used to compensate for a small deviation from equality in the allocation. But if the deviation is too big, the agents prefer to find another solution. We can check the feasibility using Inter such that the interval is the amount of money available.

The Inter variant is similar to the *Fully Polynomial-Time Approximation Scheme* (FPTAS) definition:

▶ **Definition 1.** *An* FPTAS *for MinMax-*Part$[n](\mathcal{X})$ *is an algorithm that finds, for each rational* $\epsilon > 0$*, an* n*-way partition of* $\mathcal{X}$ *with* $OPT \leq \max(b_1, \ldots, b_n) \leq (1 + \epsilon) \cdot OPT$*, where* OPT *is the smallest possible value of* $\max(b_1, \ldots, b_n)$ *in the given instance, in time* $O(poly(m, 1/\epsilon, \log S))$*.*

An FPTAS finds a solution for which the relative deviation from optimality depends on the optimal *integral* solution. In contrast, in the Dec-Inter$[n, u](\mathcal{X})$ problem, we look for a solution for which the relative deviation from optimality depends on the optimal *fractional* solution.

[1] Split the house worth 400 such that one heir gets 7/8 of it, and the other gets 1/8 of it plus both the 100 and 200 houses.

Table 1 Run-time complexity of the n-way number partitioning variants. In SplitItem, s (an integer) is the number of items the algorithm is allowed to split. In Splitting, t (an integer) is the number of splittings the algorithm is allowed to make. In Inter, u (a rational number) is the ratio between the allowed interval length and M.

Problem	Objective	Num of bins	Bound	Run-time complexity
Part	Dec	Unbounded	$s = t$	Strongly NP-hard [[8]]
		Constant n	$= u = 0$	NP-complete [[9]]
SplitItem	Dec	Unbounded	s (any)	Strongly NP-hard [[4]Corollary 19]
		Constant $n \geq 3$	$s < n - 2$	NP-complete [Theorem 14]
	MinMax		$s \geq n - 2$	$O(poly(m, \log S))$ [Theorem 16]
	MaxMin		$s \geq n - 2$	$O(poly(m, \log S))$ [[4]Theorem 22]
	All	Constant $n \geq 2$	$s \geq n - 1$	$O(m + n)$ [cut-the-line]
Splitting	Dec	Unbounded	t (any)	Strongly NP-hard [[4]Theorem 20]
		Constant n	$t < n - 1$	NP-complete [Theorem 17]
	All		$t \geq n - 1$	$O(m + n)$ [cut-the-line]
Inter	Dec	Unbounded	u (any)	Strongly NP-hard [[4]Theorem 17]
		Constant $n \geq 3$	$u < n - 2$	NP-complete [Theorem 11]
	Dec		$u \geq n - 2$	$O(poly(m, \log S))$ [Theorem 10]
		Constant $n \geq 2$	$u \geq n - 1$	$O(m + n)$ [cut-the-line+Thm 13]
		$n = 2$	$u > 0$	$O(poly(m, \log S, 1/u))$ [Theorem 6]

Contribution

When $s, t, u = 0$, SplitItem, Splitting and Inter decision versions are equivalent to the NP-hard Part decision version. In contrast, when $s, t \geq n - 1$ the problem is easily solvable by the following algorithm: put the items on a line, cut the line into n pieces with an equal total value, and put each piece in a bin. Since $n - 1$ cuts are made, at most $n - 1$ items need to be split. So for $n = 2$, the runtime complexity of the Dec-SplitItem$[n, s](\mathcal{X})$ and Dec-Splitting$[n, t](\mathcal{X})$ problem is well-understood (assuming P $\neq$ NP): it is polynomial-time solvable if and only if $s, t \geq 1$. The case for Inter is slightly different since u, v are rational numbers. We summarize all our results in Table 1.

In Section 4 we show a two-way polynomial-time reduction between problems SplitItem and Inter. This reduction is the key for many of our results. We use it to handle the case where the number of split items is smaller than $n - 2$. First, we design an FPTAS in [4][Appendix A.3]. Second, we develop a practical (not polynomial-time) algorithm, for solving MinMax-SplitItem$[n, s](\mathcal{X})$ for any $s \geq 0$. The algorithm can use any practical algorithm for solving the MinMax-Part$[n](\mathcal{X})$ problem. The latest helps us in [4][Appendix A.4] to conduct some experiences to various randomly generated instances and analyze the effect of s on the quality of the attained solution. The supplement provides complementary results and technical proof details omitted from the main text.

2 Related Work

In most combinatorial optimization problems, there is a clear distinction between discrete and continuous variables. E.g., when a problem is modeled by a mixed-integer program, each variable in the program is determined in advance to be either discrete (must get an integer value) or continuous (can get any real value). The problems we study belong to a much smaller class of problems, in which all variables are potentially continuous, but there is an upper bound on the number of variables that can be non-discrete. We describe some such problems below.

Bounded splitting in fair division. The idea of finding fair allocations with a bounded number of split items originated from [5, 6]. They presented the *Adjusted Winner* (AW) procedure for allocating items among two agents with possibly different valuations. AW finds an allocation that is *envy-free* (no agent prefers the bundle of another agent), *equitable* (both agents receive the same subjective value), and *Pareto-optimal* (there is no other allocation where some agent gains and no agent loses), and in addition, at most a single item is split between the agents. Hence, AW solves a problem that is similar to Dec-SplitItem$[n = 2, s = 1](\mathcal{X})$ but more general, since AW allows the agents to have different valuations to the same items.

Most similar to our paper is the recent work of [15]. Their goal is to find an allocation among n agents with different valuations, which is both fair and *fractionally Pareto-optimal* (fPO), a property stronger than Pareto-optimality (there is no other discrete or fractional allocation where some agent gains and no agent loses). This is a very strong requirement: when n is fixed, and the valuations are *non-degenerate* (i.e., for every two agents, no two items have the same value-ratio), the number of fPO allocations is polynomial in m, and it is possible to enumerate all such allocations in polynomial time. Based on this observation, they present an algorithm that finds an allocation with the smallest number of split items, among all allocations that are fair and fPO. In contrast, in our paper, we do not require fPO, which may allow allocations with fewer split items or splittings. However, the number of potential allocations becomes exponential, so enumerating them all is no longer feasible.

Another paper [3] studies the same problems, but, whereas our paper focuses on identical valuations, they give new results on binary valuations (i.e., each agent values each item as 0 or 1), generalized binary valuations (i.e., each agent values each item as 0 or x_i, which can be considered as the price of the item) and negative results on non-degenerate valuations, complementing the results given by [15].

Recently, [1, 2] studied an allocation problem where some items are divisible and some are indivisible. In contrast to our setting, in [1] the distinction between divisible and indivisible items is given in advance, that is, the algorithm can only divide items that are pre-determined as divisible. In [2], for each good, some agents may regard it as indivisible, while other agents may regard the good as divisible. In our setting, only the *number* of divisible items (splittings) is given in advance, but the algorithm is free to choose *which* items to split after receiving the input.

Splitting in job scheduling. There are several variants of job scheduling problems in which it is allowed to break jobs apart. They can be broadly classified into *preemption* and *splitting*. In the preemption variants, different parts of a job must be processed at different times. In the three-field notation, they are denoted by “pmtn” and were first studied by [13]. In the splitting variants, different parts of a job may be processed simultaneously on different machines. They are denoted by “split” and were introduced by [18].

Various problems have been studied in job scheduling with preemption. The most closely related to our problem is the generalized multiprocessor scheduling (GMS). It has two variants. In the first variant, the total number of preemptions is bounded by some fixed integer. In the second variant, each job j has an associated parameter that bounds the number of times j can be preempted. In both variants, the goal is to find a schedule that minimizes the makespan subject to the preemption constraints. For identical machines, [16] prove that with the bound of $n-2$ on the total number of preemptions, the problem is NP-hard, whereas [13] shows a linear-time algorithm with $n-1$ preemptions. In Theorem 17 we prove an analogous result where the bound is on the total number of splittings.

In all the works we surveyed, there is no global bound on the number of splitting jobs. As far as we know, bounding the number of splittings or split jobs was not studied before.

Fractional bin-packing. Another problem, in which splitting was studied, is the classical bin-packing problem. Bin-packing with fragmented items are first introduced by [12]. They called the problem fragmentable object bin-packing problem and prove that the problem is NP-hard. It is later split into two variants. In the first variant called bin-packing with size-increasing fragmentation (BP-SIF), each item may be fragmented; overhead units are added to the size of every fragment. In the second variant called bin-packing with size-preserving fragmentation (BP-SPF) each item has a size and a cost; fragmenting an item increases its cost but does not change its size. Menakerman and Rom [14] show that BP-SIF and BP-SPF are NP-hard in the strong sense. Despite the hardness, they present several algorithms and investigate their performance. Their algorithms use classic algorithms for bin-packing, like next-fit and first-fit decreasing, as a base for their algorithms.

Finally, the fractional knapsack problem with penalties is recently introduced by [11]. They develop an FPTAS and a dynamic program for the problem, and they show an extensive computational study comparing the performance of their models.

3 Partition with Interval Target

In this section we analyze the problems Dec-Inter$[n, v](\mathcal{X})$ and Dec-Inter$[n, u](\mathcal{X})$.

3.1 The Dec-Inter$[n, v](\mathcal{X})$ problem

Given an instance of Dec-Inter$[n, v](\mathcal{X})$, we say that a partition of $\mathcal{X}$ is *v-feasible* if $S \leq \max(b_1, \ldots, b_n) \leq (1+v) \cdot S$, where $b_1, \ldots, b_n$ are the bin sums and S is the sum of items divided by n. The Dec-Inter$[n, v](\mathcal{X})$ problem is to decide whether a v-feasible partition exists.

▶ **Lemma 2.** *When $v \geq 1$, the problem Dec-Inter$[n, v](\mathcal{X})$ can be decided in linear time by a greedy algorithm.*

The proof is in [4][Appendix B.3]. We focus below on the case $v < 1$.

▶ **Definition 3.** *Given an instance of Dec-Inter$[n, v](\mathcal{X})$, a rational number $\epsilon > 0$, and a partition of $\mathcal{X}$ among n bins, an* almost-full bin *is a bin with sum larger than $(1+v) \cdot S/(1+\epsilon)$.*

A known FPTAS for the MinMax-Part$[n](\mathcal{X})$ problem [17] gives us valuable information since we can easily verify that if the output of this FPTAS is smaller than $(1 + v) \cdot S$ then in any n-way partition of $\mathcal{X}$, at least one bin is almost-full. To gain more information on the instance, we apply an FPTAS for a constrained variant of Part, with a *Critical Coordinate*. For an integer $n \geq 2$, a list $\mathcal{X}$, and a rational number $v > 0$, we define the following problem:

MinMax-Part$[n, v, i](\mathcal{X})^2$: Minimize $\max(b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$ *subject to $b_i \leq (1 + v) \cdot S$ where $b_1, \ldots, b_n$ are bin sums in an n-way partition of $\mathcal{X}$.*

The general technique developed by [17] for converting a dynamic program to an FPTAS can be used to design an FPTAS for MinMax-Part$[n, v, i](\mathcal{X})$; we give the details in [4][Appendix C.1]. We denote by FPTAS(MinMax-Part$[n, v, i](\mathcal{X})$, ϵ) the largest bin sum in the solution obtained by the FPTAS.

▶ **Lemma 4.** *For any $n \geq 2, v > 0, \epsilon > 0$, if, for all $i \in [n]$,* FPTAS*(MinMax-Part$[n, v, i](\mathcal{X})$, ϵ) $> (1 + v) \cdot S$, then in any v-feasible n-way partition of $\mathcal{X}$, at least* two *bins are almost-full.*

[2] The critical coordinate is parameterized by i. In this work, we do not use this parameter, but other results may iterate over every critical coordinate possible.

Proof. Suppose by contradiction that there exists a v-feasible partition of $\mathcal{X}$ with at most one almost-full bin. Let i be the index of the bin with the largest sum in that partition. Since bin i has the largest sum, if there is one almost-full bin, it must be bin i. Hence, bins that are not i are not almost-full, so $\max(b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n) \leq (1+v) \cdot S/(1+\epsilon)$. Moreover, $b_i \leq (1+v) \cdot S$ since the partition is v-feasible. Therefore, FPTAS(MinMax-Part$[n, v, i](\mathcal{X})$, $\epsilon) \leq (1+v) \cdot S$ by the definition of FPTAS. This contradicts the lemma assumption. ◀

Using Theorem 4, we can now derive a complete algorithm for Dec-Inter$[n = 2, v](\mathcal{X})$.

Algorithm 1 Dec-Inter$[n = 2, v](\mathcal{X})$.

```
$b_2 \longleftarrow$ FPTAS(MinMax-Part$[n = 2, v, i](\mathcal{X})$, $\epsilon = v/2$).
If $b_2 \leq (1 + v) \cdot S$, return "yes".
Else, return "no".
```

▶ **Theorem 5.** *For any rational $v > 0$, Algorithm 1 solves the Dec-Inter$[n = 2, v](\mathcal{X})$ problem in time $O(poly(m, \log S, 1/v))$, where m is the number of items in $\mathcal{X}$ and $S = (\sum_i x_i)/n$ is the perfect bin sum.*

Proof. The run-time of Algorithm 1 is dominated by the run-time of the FPTAS for MinMax-Part$[n = 2, v, i](\mathcal{X})$, which is $O(\text{poly}(m, \log S, 1/\epsilon)) = O(\text{poly}(m, \log S, 1/v))$ (we show in [4][Appendix C.3.1] that the exact run-time is $O(\frac{m}{v} \log S)$). It remains to prove that Algorithm 1 indeed solves Dec-Inter$[n = 2, v](\mathcal{X})$ correctly.

If b_2, the returned bin sum of FPTAS(MinMax-Part$[n = 2, v, i](\mathcal{X})$, $\epsilon = v/2$), is at most $(1+v) \cdot S$, then the partition found by the FPTAS is v-feasible, so Algorithm 1 answers "yes" correctly. Otherwise, by Theorem 4, in any v-feasible partition of $\mathcal{X}$ into two bins, both bins are almost-full. This means that, in any v-feasible partition, both bin sums b_1 and b_2 are larger than $(1+v) \cdot S/(1+\epsilon)$, which is larger than S since $\epsilon = v/2$. So $b_1 + b_2 > 2S$. But this is impossible since the sum of the items is $2S$ by assumption. Hence, no v-feasible partition exists, and Algorithm 1 answers "no" correctly. [3] ◀

▶ **Corollary 6.** *For any rational $u > 0$, Algorithm 1 solves the Dec-Inter$[n = 2, u](\mathcal{X})$ problem in time $O(poly(m, \log S, 1/u))$.*

Proof. For any rational $u > 0$, let $v := uM/S$, that is $v > 0$. Algorithm 1 solves the Dec-Inter$[n = 2, v = uM/S](\mathcal{X})$ problem in time $O(\text{poly}(m, \log S, S/uM)))$, where m is the number of items in $\mathcal{X}$ and $S = (\sum_i x_i)/n$ is the perfect bin sum. Since $S \leq mM$, the algorithm runs in time $O(\text{poly}(m, \log S, 1/u))$ for any $u > 0$. ◀

▶ Remark 7. The reader may wonder why we cannot use a similar algorithm for $n \geq 3$. For example, we could have considered a variant of MinMax-Part$[n, v, i](\mathcal{X})$ with two critical coordinates:

> *Minimize* $\max(b_3, \ldots, b_n)$ *subject to* $b_1 \leq (1+v) \cdot S$ *and* $b_2 \leq (1+v) \cdot S$*, where* $b_1, b_2, b_3, \ldots, b_n$ *are bin sums in an* n*-way partition of* $\mathcal{X}$*.*

[3] Instead of an FTPAS for MinMax-Part$[n = 2, v, i](\mathcal{X})$, we could use an FTPAS for the Subset Sum problem [10], using the same arguments. The critical coordinate is not needed in the Subset Sum FPTAS, since the output is always smaller than the target. We prefer to use the FTPAS for MinMax-Part$[n = 2, v, i](\mathcal{X})$, since it is based on the general technique of [17], that we use later for solving other problems.

If the FPTAS for this problem does not find a v-feasible partition, then any v-feasible partition must have at least three almost-full bins. Since not all bins can be almost-full, one could have concluded that there is no v-feasible partition into $n = 3$ bins.

Unfortunately, the problem with two critical coordinates probably does not have an FPTAS even for $n = 3$, since it is equivalent to the Multiple Subset Sum problem, which does not have an FPTAS unless P=NP [7]. In the next subsection we handle the case $n \geq 3$ in a different way.

3.2 Dec-Inter$[n, u](\mathcal{X})$: an algorithm for $n \geq 3$ and $u \geq n - 2$

The case when $u \geq n - 1$ is solved by the cut-the-line algorithm combined with Theorem 13. Here, we prove a more general case where $u \geq n-2$. Given an instance of Dec-Inter$[n, u](\mathcal{X})$, where the sum of the items is $n \cdot S$ and the largest item is $n \cdot M$, where $S, M \in \mathbb{Q}$, we say that a partition of $\mathcal{X}$ is *u-possible* if $S \leq \max(b_1, \ldots, b_n) \leq S + u \cdot M$, where $b_1, \ldots, b_n$ are the bin sums. The Dec-Inter$[n, u](\mathcal{X})$ problem is to decide whether a u-possible partition exists. Given an instance of Dec-Inter$[n, u](\mathcal{X})$, we let $v := uM/S$, so that a partition is u-possible if and only if it is v-feasible.

The algorithm starts by running FPTAS(MinMax-Part$[n, v, i](\mathcal{X})$, $\epsilon = v/(4m^2)$). If the FPTAS find a v-feasible partition, we return "yes". Otherwise, by Theorem 4, any v-feasible partition must have at least two almost-full bins.

We take a detour from the algorithm and prove some existential results about partitions with two or more almost-full bins. We assume that there are more items than bins, that is, $m > n$. This assumption is because if $m \leq n$, one can compute all the combinations using brute force (note that the running time is polynomial since $2^m \leq 2^n = O(1)$ since n is a fixed parameter).

3.2.1 Structure of partitions with two or more almost-full bins

We distinguish between big, medium, and small items defined as follows. A *small item* is an item with length smaller than $2\epsilon nS$; a *big item* is an item with length greater than $(\frac{v}{n-2} - 2\epsilon)nS$. All other items are called *medium items.* Our main structural Lemma is the following.

▶ **Lemma 8.** *Suppose that $u \geq n-2$, $v = uM/S < 1$, $\epsilon = v/4m^2$ and the following properties hold.*

(1) *There is no v-feasible partition with at most 1 almost-full bin;*
(2) *There is a v-feasible partition with at least 2 almost-full bins.*

Then, there is a v-feasible partition with the following properties.

(a) *Exactly two bins (w.l.o.g. bins 1 and 2) are almost-full.*
(b) *The sum of every not-almost-full bin $i \in \{3, \ldots, n\}$ satisfies*

$$\left(1 - \frac{2}{n-2}v - 2\epsilon\right) \cdot S \leq b_i \leq \left(1 - \frac{2}{n-2}v + (n-1)2\epsilon\right) \cdot S.$$

(c) *Every item in an almost-full bin is a big-item.*
(d) *Every item in a not-almost-full bin is either a small-item, or a big-item larger or equal to every item in bins 1,2.*
(e) *There are no* medium-items *at all.*
(f) *Every not-almost-full bin contains the same number of big-items, say ℓ, where ℓ is an integer (it may contain, in addition, any number of small-items).*
(g) *Every almost-full bin contains $\ell + 1$ big-items (and no small-items).*

As an example of this situation, consider an instance with 7 items, all of which have size 1, with $n = 5$ and $u = 3$. Then, there is a u-possible partition with two almost-full bins: $(1, 1), (1, 1), (1), (1), (1)$, and no u-possible partition with 1 or 0 almost-full bins. See [4][Appendix B.4.1] for details. A full proof ([4][Appendix B.4.2]) of the Lemma appears in the appendix, here we provide a sketch proof.

Proof Sketch. We start with an arbitrary v-feasible partition with some $r \geq 2$ almost-full bins $1, \ldots, r$, and convert it using a sequence of transformations to another v-feasible partition satisfying properties (a)–(g), as explained below. Note that the transformations are not part of our algorithm and are only used to prove the lemma. First, we note that there must be at least one bin that is not almost-full, since the sum of an almost-full bin is larger than S whereas the sum of all n bins is $n \cdot S$.

For (a), if there are $r \geq 3$ almost-full bins, we move any item from one of the almost-full bins $3, \ldots, r$ to some not-almost-full bin. We prove that, as long as $r \geq 3$, the target bin remains not-almost-full. This transformation is repeated until $r = 2$ and only bins 1 and 2 remain almost-full.

For (b), for the lower bound, if there is $i \in \{3, \ldots, n\}$ for which b_i is smaller than the lower bound, we move an item from bins $1, 2$ to bin i. We prove that bin i remains not-almost-full, so by assumption (1), bins 1, 2 must remain almost-full. We repeat until b_i satisfies the lower bound. Once all bins satisfy the lower bound, we prove that the upper bound is satisfied too.

For (c), if bin 1 or 2 contains an item that is not big, we move it to some bin $i \in \{3, \ldots, n\}$. We prove that bin i remains not-almost-full, so by assumption (1), bins 1, 2 must remain almost-full. We repeat until bins 1 and 2 contain only big-items.

For (d), if some bin $i \in \{3, \ldots, n\}$ contains an item bigger than $2nS\epsilon$ and smaller than any item in bin 1 or bin 2, we exchange it with an item from bin 1 or 2. We prove that, after the exchange, b_i remains not-almost-full, so bins 1, 2 must remain almost-full. We repeat until bins 1,2 contain only the smallest big-items. Note that transformations (b), (c), (d) increase the sum in the not-almost-full bins $3, \ldots, n$, so the process must end.

For (e), it follows logically from properties (d) and (c): if bins 1,2 contain only big items and the other bins contain only big and small items, then the instance cannot contain any medium items (that are neither big nor small). For clarity and verification, we provide a stand-alone proof.

For (f), we use the fact that the difference between two not-almost-full bins is at most $2nS\epsilon$ by property (b), and show that it is too small to allow a difference of a whole big-item.

For (g), because by (d) bins 1 and 2 contain the smallest big-items, whereas their sum is larger than bins $3, \ldots, n$, they must contain at least $\ell + 1$ big-items. We prove that, if they contain $\ell + 2$ big-items, then their sum is larger than $(1 + v)S$, which contradicts v-feasibility. ◀

Properties (f) and (g) imply:

▶ **Corollary 9.** *Suppose that $u \geq n - 2$, $v = uM/S$ and $\epsilon = v/4m^2$. Let $B \subseteq \mathcal{X}$ be the set of big items in $\mathcal{X}$. If there is a v-feasible partition with at least two almost-full bins, and no v-feasible partition with at most one almost-full bin, then $|B| = n\ell + 2$ for $\ell \in \mathbb{N}$.*

3.2.2 Back to the algorithm

We have left the algorithm at the point when FPTAS(MinMax-Part$[n=2,v,i](\mathcal{X})$, $\epsilon = v/4m^2) > (1+v)\cdot S$ that is the FPTAS did not return a v-feasible partition. Theorem 4 implies that if a v-feasible partition exists, then there exists a v-feasible partition satisfying all properties of Theorem 8 and Theorem 9. We can find such a partition (if it exists) in two steps:

- **For bins** $1,2$**:** Find a v-feasible partition of the $2\ell+2$ smallest items in B into two bins with $\ell+1$ items in each bin.
- **For bins** $3,\dots,n$**:** Find a v-feasible partition of the remaining items in $\mathcal{X}$ into $n-2$ bins.

For bins $3,\dots,n$, we use the FPTAS for the problem MinMax-Part$[n=n-2](\mathcal{X})$. If it returns a v-feasible partition, we are done. Otherwise, by FPTAS definition, every partition into $(n-2)$ bins must have at least one almost-full bin. But by Theorem 8(a), all bins $3,\dots,n$ are not almost-full which is a contradiction. Therefore, if the FPTAS does not find a v-feasible partition, we answer “no”. Bins 1 and 2 require a more complicated algorithm that is explained in [4][Appendix B.2]. We are now ready to present the complete algorithm for Dec-Inter$[n,u](\mathcal{X})$, presented in Algorithm 2.

Algorithm 2 Dec-Inter$[n,u](\mathcal{X})$ (complete algorithm).

```
1:  $v \longleftarrow uM/S$ and $\epsilon \longleftarrow v/(4m^2)$.
2:    If FPTAS(MinMax-Part$[n,v,i](\mathcal{X})$, $\epsilon$) $\leq (1+v)\cdot S$, return “yes”.
3:  $B \longleftarrow \left\{x_i \in \mathcal{X} \mid x_i > nS(\frac{v}{n-2} - 2\epsilon)\right\}$          ▷ big items
4:    If $|B|$ is not of the form $n\ell+2$ for some integer $\ell$, return “no”.
5:  $B_{1:2} \longleftarrow$ the $2\ell+2$ smallest items in $B$.          ▷ break ties arbitrarily
6:  $B_{3..n} \longleftarrow \mathcal{X} \setminus B_{1:2}$.          ▷ big and small items
7:    $b_3 \longleftarrow$ FPTAS(MinMax-Part$[n=n-2](B_{3..n})$, $\epsilon$)   ▷ Computes an approximately-optimal
    $(n-2)-$way partition of $B_{3..n}$ and returns the maximum bin sum in the partition.
8:  If $b_3 > (1+v)S$, return “no”.          ▷ The FTPAS did not find a $v$-feasible partition.
9:    $\overline{B_{1:2}} \longleftarrow \{nM_{1:2} - x \mid x \in B_{1:2}\}$ and $\overline{v} \longleftarrow (S+vS-S_{1:2})/\overline{S_{1:2}}$.
10: Look for a $\overline{v}$-feasible partition of $\overline{B_{1:2}}$ into two subsets of $\ell+1$ items (see [4][Appendix
    B.2]).
11: If a $\overline{v}$-feasible partition is found, return “yes”. Else, return “no”.
```

▶ **Theorem 10.** *For any fixed integer $n \geq 3$ and rational number $u \geq n-2$, Algorithm 2 solves Dec-Inter$[n,u](\mathcal{X})$ in $O(poly(m,\log S))$ time, where m is the number of items in $\mathcal{X}$, and S is the average bin size.*

Proof. If Algorithm 2 answers “yes”, then clearly a v-feasible partition exists. To complete the correctness proof, we have to show that the opposite is true as well.

Suppose there exists a v-feasible partition. If the partition has at most one almost-full bin, then by Theorem 4, it is found by the FPTAS in step 2. Otherwise, the partition must have at least two almost-full bins, and there exists a v-feasible partition satisfying the properties of Theorem 8. By Theorem 9, the algorithm does not return “no” in step 4. By properties (a) and (b), there exists a partition of $B_{3..n}$ into $n-2$ bins $3,\dots,n$ which are not almost-full. By definition, the FPTAS in step 7 finds a partition with $\max(b_3,\dots,b_n) \leq (1+v)S$. The final steps, regarding the partition of $B_{1:2}$, are justified by the discussion at [4][Appendix B.2]. The complete running time $O(\text{poly}(m,\log S))$ of Algorithm 2 is justified by the running time of the FPTAS for FPTAS(MinMax-Part$[n=2,v,i](\mathcal{X})$, ϵ)

and for FPTAS(MinMax-Part$[n = n-2](B_{3..n}), \epsilon)$. Note that $1/v$ is polynomial in m since $1/v = S/uM \leq mM/uM = m/u = O(m)$ since u is fixed. The exact running time, $O(m^4 \log S)$, is detailed in [4][Appendix C.3.2]. ◀

3.3 Hardness for $n \geq 3$ bins and $u < n-2$

The following theorem complements the previous subsection.

▶ **Theorem 11.** *Given a fixed integer $n \geq 3$ and a positive rational number $u < n-2$, the problem Dec-Inter$[n,u](\mathcal{X})$ is* NP*-complete.*

Proof. Given an n-way partition of m items, summing the sizes of all elements in each bin allows us to check whether the partition is u-possible in linear time. So, the problem is in NP. To prove that Dec-Inter$[n,u](\mathcal{X})$ is NP-Hard, we reduce from the equal-cardinality partition problem, proved to be NP-hard in [9]: given a list with an even number of integers, decide if they can be partitioned into two subsets with the same sum and the same cardinality.

Given an instance $\mathcal{X}_1$ of equal-cardinality partition, denote the number of items in $\mathcal{X}_1$ by $2m'$. Define M to be the sum of numbers in $\mathcal{X}_1$ divided by $2n(1-\frac{u}{n-2})$, so that the sum of items in $\mathcal{X}_1$ is $2n(1-\frac{u}{n-2})M$ (where n and u are the parameters in the theorem statement). We can assume w.l.o.g. that all items in $\mathcal{X}_1$ are at most $n(1-\frac{u}{n-2})M$, since if some item is larger than half of the sum, the answer is necessarily "no".

Construct an instance $\mathcal{X}_2$ of the equal-cardinality partition problem by replacing each item x in $\mathcal{X}_1$ by $nM - x$. So $\mathcal{X}_2$ contains $2m'$ items between $n(\frac{u}{n-2})M$ and nM. Their sum, which we denote by $2S'$, satisfies $2S' = 2m' \cdot nM - 2n\left(1-\frac{u}{n-2}\right)M = 2n\left(m'-1+\frac{u}{n-2}\right)M$. Clearly, $\mathcal{X}_1$ has an equal-sum equal-cardinality partition (with bin sums $n\left(1-\frac{u}{n-2}\right)M$) iff $\mathcal{X}_2$ has an equal-sum equal-cardinality partition (with bin sums $S' = n\left(m'-1+\frac{u}{n-2}\right)M$).

Construct an instance $(\mathcal{X}_3, u)$ of Dec-Inter$[n,u](\mathcal{X})$ by adding $(n-2)(m'-1)$ items of size nM. Note that nM is indeed the largest item size in $\mathcal{X}_3$. Denote the sum of item sizes in $\mathcal{X}_3$ by nS. Then

$$\begin{aligned} nS &= 2S' + (n-2)(m'-1)\cdot nM = n\left(2(m'-1) + \frac{2u}{n-2} + (n-2)(m'-1)\right)\cdot M \\ &= n\left(n(m'-1) + \frac{2u}{n-2}\right)M; \end{aligned}$$

$$S + uM = \left(n(m'-1) + \frac{2u}{n-2} + u\right)M = \left(n(m'-1) + \frac{nu}{n-2}\right)M = S',$$

so a partition of $\mathcal{X}_3$ is u-possible if and only if the sum of each of the n bins in the partition is at most $S + uM = S'$.

We now prove that if $\mathcal{X}_2$ has an equal-sum equal-cardinality partition, then the instance $(\mathcal{X}_3, u)$ has a u-possible partition, and vice versa. If $\mathcal{X}_2$ has an equal-sum partition, then the items of $\mathcal{X}_2$ can be partitioned into two bins of sum S', and the additional $(n-2)(m'-1)$ items can be divided into $n-2$ bins of $m'-1$ items each. The sum of these items is

$$(m'-1)\cdot nM = n(m'-1)M = S - \frac{2}{n-2}uM < S + uM = S', \tag{1}$$

so the resulting partition is a u-possible partition of $\mathcal{X}_3$. Conversely, suppose $\mathcal{X}_3$ has a u-possible partition. Let us analyze its structure.

Since the partition is u-possible, the sum of every two bins is at most $2(S + uM)$. So the sum of every $n - 2$ bins is at least $nS - 2(S + uM) = (n - 2)S - 2uM$. Since the largest $(n - 2)(m' - 1)$ items in $\mathcal{X}_3$ sum up to exactly $(n - 2)S - 2uM$ by (1), every $n - 2$ bins must contain at least $(n - 2)(m' - 1)$ items. Since $\mathcal{X}_3$ has $(n - 2)(m' - 1) + 2m'$ items overall, $n - 2$ bins must contain exactly $(n - 2)(m' - 1)$ items, such that each item size must be nM, and their sum must be $(n - 2)S - 2uM$. The other two bins contain together $2m'$ items with a sum of $2(S + uM)$, so each of these bins must have a sum of exactly $S + uM$. Since $(m' - 1) \cdot nM < S + uM$ by (1), each of these two bins must contain exactly m' items. These latter two bins are an equal-sum equal-cardinality partition for $\mathcal{X}_2$. This construction is done in polynomial time, completing the reduction. ◀

4 Partition with Split Items

We now deal with the problem SplitItem. We redefine the Dec-SplitItem$[n, s](\mathcal{X})$ problem. For a fixed number $n \geq 2$ of bins, given a list $\mathcal{X}$, the number of split items $s \in \{0, \dots, m\}$ and a rational number $v \geq 0$, define:

Dec-SplitItem$[n, s, v](\mathcal{X})$: Decide if there exists a partition of $\mathcal{X}$ among n bins with at most s split items, such that $\max(b_1, \dots, b_n) \leq (1 + v)S$.

The special case $v = 0$ corresponds to the Dec-SplitItem$[n, s](\mathcal{X})$ problem. The following Lemma shows that, w.l.o.g., we can consider only the longest items for splitting.

▶ **Lemma 12.** *For every partition with $s \in \mathbb{N}$ split items and bin sums $b_1, \dots, b_n$, there exists a partition with the same bin sums $b_1, \dots, b_n$ in which only the s* largest *items are split.*

Proof. Consider a partition in which some item with length x is split between two or more bins, whereas some item with length $y > x$ is allocated entirely to some bin i. Construct a new partition as follows: first move item x to bin i; second remove from bin i, a fraction $\frac{x}{y}$ of item y; and finally split that fraction of item y among the other bins, in the same proportions as the previous split of item x. All bin sums remain the same. Repeat the argument until only the longest items are split. ◀

▶ **Theorem 13.** *For any fixed integers $n \geq 2$ and $u \geq 0$, there is a polynomial-time reduction from Dec-Inter$[n, u](\mathcal{X})$ to Dec-SplitItem$[n, s = u, v = 0](\mathcal{X})$.*

Proof. Given an instance $\mathcal{X}$ of Dec-Inter$[n, u](\mathcal{X})$, we add u items of size nM, where nM is the size of the biggest item in $\mathcal{X}$ to construct an instance $\mathcal{X}'$ of Dec-SplitItem$[n, s = u, v = 0](\mathcal{X}')$.

First, assume that $\mathcal{X}$ has a u-possible partition. Then there are n bins with a sum at most $S + uM$. Take the u added items of size nM and add them to the bins, possibly splitting some items between bins, such that the sum of each bin becomes exactly $S + uM$. This is possible because the sum of the items in $\mathcal{X}'$ is $nS + unM = n(S + uM)$. The result is a 0-feasible partition of $\mathcal{X}'$ with at most u split items.

Second, assume that $\mathcal{X}'$ has a 0-feasible partition with at most u split items. Then there are n bins with a sum of exactly $S + uM$. By Theorem 12, we can assume the split items are the largest ones, which are the u added items of size nM. Remove these items to get a partition of $\mathcal{X}$. The sum in each bin is now at most $S + uM$, so the partition is u-possible. This construction is done in polynomial time, which completes the proof. ◀

▶ **Corollary 14.** *For every fixed integers $n \geq 3$ and $s \in \{0, \dots, n - 3\}$, the problem Dec-SplitItem$[n, s](\mathcal{X})$ is* NP*-complete.*

Proof. Theorem 11 and Theorem 13 imply that Dec-SplitItem$[n,s](\mathcal{X})$ is NP-hard. The problem Dec-SplitItem$[n,s](\mathcal{X})$ is in NP since given a partition, summing the sizes of the items (or items fractions) in each bin let us check in linear time whether the partition has equal bin sums. ◀

▶ **Theorem 15.** *For any fixed integers $n \geq 2, s \geq 0$ and rational $v \geq 0$, there is a polynomial-time reduction from Dec-SplitItem$[n,s,v](\mathcal{X})$, to Dec-Inter$[n,u](\mathcal{X})$ for some rational number $u \geq s$.*

Proof. Given an instance $\mathcal{X}$ of Dec-SplitItem$[n,s,v](\mathcal{X})$, denote the sum of all items in $\mathcal{X}$ by nS and the largest item size by nM where $S, M \in \mathbb{Q}$. Construct an instance $\mathcal{X}'$ of Dec-Inter$[n,u](\mathcal{X}')$ by removing the s largest items from $\mathcal{X}$. Denote the sum of remaining items by nS' for some $S' \leq S$, and the largest remaining item size by nM' for some $M' \leq M$. Note that the size of every removed item is between nM' and nM, so $sM' \leq S - S' \leq sM$. Set $u := (S + vS - S')/M'$, so $S' + uM' = S + vS$. Note that $u \geq (S - S')/M' \geq s$.

First, assume that $\mathcal{X}$ has a v-feasible partition with s split items. By Theorem 12, we can assume that only the s largest items are split. Therefore, removing the s largest items results in a partition of $\mathcal{X}'$ with no split items, where the sum in each bin is at most $S + vS = S' + uM'$. This is a u-possible partition of $\mathcal{X}'$.

Second, assume that $\mathcal{X}'$ has a u-possible partition. In this partition, each bin sum is at most $S' + uM' = S + vS$, so it is a v-feasible partition of $\mathcal{X}'$. To get a v-feasible partition of $\mathcal{X}$, take the s previously removed items and add them to the bins, possibly splitting some items between bins, such that the sum in each bin remains at most $S + vS$. This is possible since the items sum is $nS \leq n(S + vS)$. This construction is done in polynomial time. ◀

Combining Theorem 15 with Theorem 10 provides a polynomial time algorithm to solve Dec-SplitItem$[n,s,v](\mathcal{X})$ for any fixed $n \geq 3, s \geq n-2$ and rational $v \geq 0$. The latter is used to solve the MinMax-SplitItem$[n,s](\mathcal{X})$ optimization problem by using binary search on the parameter v of the Dec-SplitItem$[n,s,v](\mathcal{X})$ problem. The details are given in [4][Appendix B.1]. The binary search procedure needs to solve at most $\log_2(nS)$ instances of Dec-SplitItem$[n,s,v](\mathcal{X})$.

▶ **Corollary 16.** *For any fixed integers $n \geq 3$ and $s \geq n-2$, MinMax-SplitItem$[n,s](\mathcal{X})$ can be solved in $O(poly(m, \log S))$ time.*

We complete this result by providing a polynomial-time algorithm for the max-min version: MaxMin-SplitItem$[n,s](\mathcal{X})$ for $s \geq n-2$ in [4][Appendix A.1].

5 Partition with Splittings

In this section, we analyze the Splitting variant.

▶ **Theorem 17.** *For any fixed integer $n \geq 2$ and fixed $t \in \mathbb{N}$ such that $t \leq n-2$, the problem Dec-Splitting$[n,t](\mathcal{X})$ is NP-complete.*

Proof. Given a partition with n bins, m items, and t splittings, summing the size of each item (or fraction of item) in each bin allows us to check whether or not the partition is perfect in linear time. So, the problem is in NP.

To prove that Dec-Splitting$[n,t](\mathcal{X})$ is NP-Hard, we apply a reduction from the Subset Sum problem. We are given an instance $\mathcal{X}_1$ of Subset Sum with m items summing up to S and target sum $T < S$. We build an instance $\mathcal{X}_2$ of Dec-Splitting$[n,t](\mathcal{X}_2)$ by adding two items, x_1, x_2, such that $x_1 = S + T$ and $x_2 = 2S(t+1) - T$ and $n - 2 - t$ auxiliary items of size $2S$. Notice that the sum of the items in $\mathcal{X}_2$ equals

$$S + (S + T) + 2S(t + 1) - T + 2S(n - 2 - t) = 2S + 2S(t + 1) + 2S(n - 2 - t)$$
$$= 2S \cdot (1 + t + 1 + n - 2 - t) = 2Sn.$$

The goal is to partition items into n bins with a sum of $2S$ per bin, and at most t splittings.

First, assume that there is a subset of items W_1 in $\mathcal{X}_1$ with a sum equal to T. Define a set, W_2, of items that contains all items in $\mathcal{X}_1$ that are not in W_1, plus x_1. The sum of W_2 is $(S - T) + x_1 = S + T + S - T = 2S$. Assign the items of W_2 to the first bin. Assign each auxiliary item to a different bin. There are $n - (n - 2 - t + 1) = t + 1$ bins left. The sum of the remaining items is $2S(t + 1)$. Using the "cut-the-line" algorithm described in the introduction, these items can be partitioned into $t + 1$ bins of equal sum $2S$, with at most t splittings. All in all, there are n bins with a sum of $2S$ per bin, and the total number of splittings is at most t.

Second, assume that there exists an equal partition for n bins with t splittings. Since $x_2 = 2S(t + 1) - T = 2S \cdot t + (2S - T) > 2S \cdot t$, this item must be split between $t + 1$ bins, which makes the total number of splittings at least t. Also, the auxiliary items must be assigned without splittings into $n - 2 - t$ different bins. There is $n - t - 1 - n + 2 + t = 1$ bin remaining, say bin i, containing only whole items, not containing any part of x_2, and not containing any auxiliary item. Bin i must contain x_1, otherwise its sum is at most S (sum of items in $\mathcal{X}_1$). Let W_1 be the items of $\mathcal{X}_1$ that are not in bin i. The sum of W_1 is $S - (2S - x_1) = x_1 - S = T$, so it is a solution to $\mathcal{X}_1$. ◀

6 Conclusion and Future Directions

We presented three variants of the n-way number partitioning problem.

In the language of fair item allocation, we have solved the problem of finding a fair allocation among n agents with identical valuations, when the ownership of some s items may be split between agents. When agents may have different valuations, there are various fairness notions, such as *proportionality*, *envy-freeness* or *equitability*. A future research direction is to develop algorithms for finding such allocations with a bounded number of shared items. We already have preliminary results for proportional allocation among three agents with different valuations, which are based on the algorithms in the present paper.

In the language of machine scheduling, MinMax-SplitItem$[n, s](\mathcal{X})$ corresponds to finding a schedule that minimizes the makespan on n identical machines when s jobs can be split between the machines; Dec-Inter$[n, u](\mathcal{X})$ corresponds to find a schedule in which the makespan is in a given interval. In a separate technical report, we have replicated the results in the present paper for the more general case of *uniform machines* in which machines may have different speeds r_j, such that a job with length x_i runs on machine j in time x_i/r_j. It may be interesting to study the more general setting of *unrelated machines*.

Our analysis shows the similarities and differences between these variants and the more common notion of FPTAS. One may view our results as introducing a new kind of approximation that approximates a decision problem by returning "yes" if and only if there exists a solution between PER and $(1 + v) \cdot PER$, where PER represents the value of a perfect solution. For the n-way number partitioning problem, a perfect solution is easy to define: it is a partition with equal bin sums. A more general definition of PER could be the solution to the fractional relaxation of an integer linear program representing the problem. As shown, NP-hard decision problems may become tractable when v is sufficiently large.

References

1 Xiaohui Bei, Zihao Li, Jinyan Liu, Shengxin Liu, and Xinhang Lu. Fair division of mixed divisible and indivisible goods. *Artif. Intell.*, 293:103436, 2021. doi:10.1016/J.ARTINT.2020.103436.

2 Xiaohui Bei, Shengxin Liu, and Xinhang Lu. Fair division with subjective divisibility. *CoRR*, abs/2310.00976, 2023. doi:10.48550/arXiv.2310.00976.

3 Samuel Bismuth, Ivan Bliznets, and Erel Segal-Halevi. Fair division with bounded sharing: Binary and non-degenerate valuations. In Guido Schäfer and Carmine Ventre, editors, *Algorithmic Game Theory*, pages 89–107, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-71033-9_6.

4 Samuel Bismuth, Vladislav Makarov, Erel Segal-Halevi, and Dana Shapira. Partitioning problems with splittings and interval targets, 2024. arXiv:2204.11753.

5 Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake Cutting to Dispute Resolution.* Cambridge University Press, Cambridge UK, February 1996.

6 Steven J. Brams and Alan D. Taylor. *The win-win solution - guaranteeing fair shares to everybody.* W. W. Norton & Company, New York, 2000.

7 Alberto Caprara, Hans Kellerer, and Ulrich Pferschy. The multiple subset sum problem. *SIAM Journal on Optimization*, 11(2):308–319, 2000. doi:10.1137/S1052623498348481.

8 M. R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4(4):397–411, 1975. doi:10.1137/0204035.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

10 Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences*, 66(2):349–370, 2003. doi:10.1016/S0022-0000(03)00006-0.

11 Enrico Malaguti, Michele Monaci, Paolo Paronuzzi, and Ulrich Pferschy. Integer optimization with penalized fractional values: The knapsack case. *Eur. J. Oper. Res.*, 273(3):874–888, 2019. doi:10.1016/j.ejor.2018.09.020.

12 C.A. Mandal, P.P. Chakrabarti, and S. Ghose. Complexity of fragmentable object bin packing and an application. *Computers and Mathematics with Applications*, 35(11):91–97, 1998. doi:10.1016/S0898-1221(98)00087-X.

13 Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959. URL: http://www.jstor.org/stable/2627472.

14 Nir Menakerman and Raphael Rom. Bin packing with item fragmentation. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 7th International Workshop, WADS 2001, Providence, RI, USA, August 8-10, 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2001. doi:10.1007/3-540-44634-6_29.

15 Fedor Sandomirskiy and Erel Segal-Halevi. Efficient fair division with minimal sharing. *Oper. Res.*, 70(3):1762–1782, 2022. doi:10.1287/opre.2022.2279.

16 Evgeny Shchepin and Nodari Vakhania. New tight np-hardness of preemptive multiprocessor and open-shop scheduling. In *Proceedings of 2nd multidisciplinary international conference on scheduling: theory and applications MISTA 2005*, pages 606–629, 2005.

17 Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS J. Comput.*, 12(1):57–74, 2000. doi:10.1287/IJOC.12.1.57.11901.

18 Wenxun Xing and Jiawei Zhang. Parallel machine scheduling with splitting jobs. *Discret. Appl. Math.*, 103(1-3):259–269, 2000. doi:10.1016/S0166-218X(00)00176-1.

The Existential Theory of the Reals with Summation Operators

Markus Bläser
Saarland University, Saarbrücken, Germany

Julian Dörfler
Saarland University, Saarbrücken, Germany

Maciej Liśkiewicz
University of Lübeck, Germany

Benito van der Zander
University of Lübeck, Germany

Abstract

To characterize the computational complexity of satisfiability problems for probabilistic and causal reasoning within Pearl's Causal Hierarchy, van der Zander, Bläser, and Liśkiewicz [IJCAI 2023] introduce a new natural class, named succ-$\exists\mathbb{R}$. This class can be viewed as a succinct variant of the well-studied class $\exists\mathbb{R}$ based on the Existential Theory of the Reals (ETR). Analogously to $\exists\mathbb{R}$, succ-$\exists\mathbb{R}$ is an intermediate class between NEXP and EXPSPACE, the exponential versions of NP and PSPACE.

The main contributions of this work are threefold. Firstly, we characterize the class succ-$\exists\mathbb{R}$ in terms of nondeterministic real Random-Access Machines (RAMs) and develop structural complexity theoretic results for real RAMs, including translation and hierarchy theorems. Notably, we demonstrate the separation of $\exists\mathbb{R}$ and succ-$\exists\mathbb{R}$. Secondly, we examine the complexity of model checking and satisfiability of fragments of existential second-order logic and probabilistic independence logic. We show succ-$\exists\mathbb{R}$-completeness of several of these problems, for which the best-known complexity lower and upper bounds were previously NEXP-hardness and EXPSPACE, respectively. Thirdly, while succ-$\exists\mathbb{R}$ is characterized in terms of ordinary (non-succinct) ETR instances enriched by exponential sums and a mechanism to index exponentially many variables, in this paper, we prove that when only exponential sums are added, the corresponding class $\exists\mathbb{R}^{\Sigma}$ is contained in PSPACE. We conjecture that this inclusion is strict, as this class is equivalent to adding a VNP-oracle to a polynomial time nondeterministic real RAM. Conversely, the addition of exponential products to ETR, yields PSPACE. Furthermore, we study the satisfiability problem for probabilistic reasoning, with the additional requirement of a small model, and prove that this problem is complete for $\exists\mathbb{R}^{\Sigma}$.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Existential theory of the real numbers, Computational complexity, Probabilistic logic, Models of computation, Existential second order logic

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.13

Related Version *Full Version*: https://arxiv.org/abs/2405.04697

Funding *Benito van der Zander*: Work supported by the Deutsche Forschungsgemeinschaft (DFG) grant 471183316 (ZA 1244/1-1).

1 Introduction

The existential theory of the reals, ETR, is the set of all true sentences of the form

$$\exists x_0 \dots \exists x_n \; \varphi(x_0, \dots, x_n), \tag{1}$$

where φ is a quantifier-free Boolean formula over the basis $\{\vee, \wedge, \neg\}$, variables $x_0, \dots, x_n$, and a signature consisting of the constants 0 and 1, the functional symbols $+$ and $\cdot$, and the

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 13; pp. 13:1–13:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

relational symbols $<$, $\leq$, and $=$. The sentences are interpreted over the real numbers in the standard way. The significance of this theory lies in its exceptional expressiveness, enabling the representation of numerous natural problems across computational geometry [1, 20, 8], Machine Learning and Artificial Intelligence [2, 21, 31], game theory [4, 12], and various other domains. Consequently, a complexity class, $\exists\mathbb{R}$, has been introduced to capture the computational complexity associated with determining the truth within the existential theory of the reals. This class is formally defined as the closure of ETR under polynomial-time many-one reductions [14, 7, 25]. For a comprehensive compendium on $\exists\mathbb{R}$, see [26].

Our study focuses on ETR which extends the syntax of formulas to allow the use of summation operators in addition to the functional symbols $+$ and $\cdot$. This research direction, initiated in [31], was motivated by an attempt to accurately characterize the computational complexity of satisfiability problems for probabilistic and causal reasoning across "Pearl's Causal Hierarchy" (PCH) [28, 23, 3].

In [31], the authors introduce a new natural class, named $\mathsf{succ}\text{-}\exists\mathbb{R}$, which can be viewed as a succinct variant of $\exists\mathbb{R}$. Perhaps, one of the most notable complete problems for the new class is the problem, called Σ_{vi}-ETR ("*vi*" stands for variable indexing). It is defined as an extension of ETR by adding to the signature an additional summation operator[1] $\sum_{x_j=0}^{1}$ which can be used to *index* the quantified variables x_i used in Formula (1). To this end, the authors define variables of the form $x_{\langle x_{j_1},\dots,x_{j_m}\rangle}$, which represent indexed variables with the index given by $x_{j_1},\dots,x_{j_m}$ interpreted as a number in binary. They can only be used when variables $x_{j_1},\dots,x_{j_m}$ occur in the scope of summation operators with range $\{0,1\}$. E.g., $\exists x_0 \dots \exists x_{2^N-1} \sum_{e_1=0}^{1} \cdots \sum_{e_N=0}^{1} (x_{\langle e_1,\dots,e_N\rangle})^2 = 1$ is a Σ_{vi}-ETR sentence[2] encoding a unit vector in $\mathbb{R}^{2^N}$. Note that sentences of Σ_{vi}-ETR allow the use of exponentially many variables. Another example sentence is $\sum_{x_1=0}^{1}\sum_{x_2=0}^{1}(x_1+x_2)(x_1+(1-x_2))(1-x_1)=0$ that models the co-SAT instance $(p \vee q) \wedge (p \vee \overline{q}) \wedge \overline{p}$. It shows that the summation operator can also be used in Σ_{vi}-ETR formulas in a standard way.

Analogously to $\exists\mathbb{R}$, $\mathsf{succ}\text{-}\exists\mathbb{R}$ is an intermediate class between the exponential versions of NP and PSPACE:

$$\mathsf{NP} \subseteq \exists\mathbb{R} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NEXP} \subseteq \mathsf{succ}\text{-}\exists\mathbb{R} \subseteq \mathsf{EXPSPACE}. \tag{2}$$

An interesting challenge, in view of the new class, is to determine whether it contains harder problems than NEXP and to examine the usefulness of $\mathsf{succ}\text{-}\exists\mathbb{R}$-completeness as a tool for understanding the apparent intractability of natural problems. A step in these directions, that we take in this work, is to express $\mathsf{succ}\text{-}\exists\mathbb{R}$ in terms of machine models over the reals, which in the case of $\exists\mathbb{R}$ yield an elegant and useful characterization by $\mathsf{NP}_{\text{real}}$ [10].

In our work, we study also the different restrictions on which the summation operators in ETR are allowed to be used and the computational complexity of deciding the resulting problems. In particular, we investigate $\exists\mathbb{R}^{\Sigma}$ – the class based on ETR enriched with standard summation operator, and $\mathsf{succ}\text{-}\exists\mathbb{R}_{\text{poly}}$ which is based on Σ_{vi}-ETR with the restriction that only polynomially many variables can be used.

In this paper, we employ a family of satisfiability problems for probabilistic reasoning, which nicely demonstrates the expressiveness of the ETR variants under consideration and illustrates the natural necessity of introducing the summation operator.

[1] In [31], the authors assume arbitrary integer lower and upper bound in $\sum_{x_j=a}^{b}$. It is easy to see that, w.l.o.g., one can restrict a and b to binary values.

[2] We represent the instances in Σ_{vi}-ETR omitting the (redundant) block of existential quantifiers, so the encoding of the example instance has length polynomial in N.

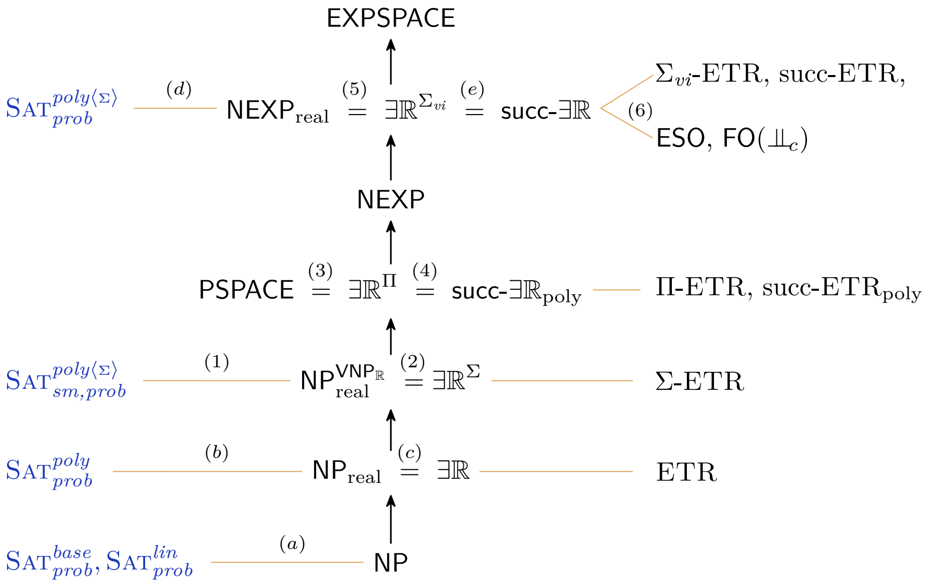

EXPSPACE
$\textsc{Sat}^{poly\langle\Sigma\rangle}_{prob}$
(d)
$\mathsf{NEXP}_{\text{real}}$
(5)
$\exists\mathbb{R}^{\Sigma_{vi}}$
(e)
succ-$\exists\mathbb{R}$
(6)
Σ_{vi}-ETR, succ-ETR,
ESO, FO($\perp\!\!\!\perp_c$)
NEXP
PSPACE
(3)
$\exists\mathbb{R}^{\Pi}$
(4)
succ-$\exists\mathbb{R}_{\text{poly}}$
Π-ETR, succ-ETR$_{\text{poly}}$
$\textsc{Sat}^{poly\langle\Sigma\rangle}_{sm,prob}$
(1)
$\mathsf{NP}^{\mathsf{VNP}_{\mathbb{R}}}_{\text{real}}$
(2)
$\exists\mathbb{R}^{\Sigma}$
Σ-ETR
$\textsc{Sat}^{poly}_{prob}$
(b)
$\mathsf{NP}_{\text{real}}$
(c)
$\exists\mathbb{R}$
ETR
$\textsc{Sat}^{base}_{prob}$, $\textsc{Sat}^{lin}_{prob}$
(a)
NP

Figure 1 The landscape of complexity classes of the existential theories of the reals and the satisfiability problems for the probabilistic languages (to the left-hand side) complete for the corresponding classes. Arrows "→" denote inclusions ⊆ and the earth-yellow labeled lines "—" connect complexity classes with complete problems for those classes. The completeness results (a), (b), and (d) were proven by Fagin et al. [11], Mossé et al. [21], resp. van der Zander et al. [31]. The characterization (c) is due to Erickson et al. [10] and (e) is proven in [31]. The references to our results are as follows: (1) Theorem 26, (2) Theorem 24, (3) and (4) Theorem 19, (5) Lemma 2, and (6) Theorem 12.

Related Work to our Study

In a pioneering paper in this field, Fagin, Halpern, and Megiddo [11] consider the probabilistic language consisting of Boolean combinations of (in)equalities of *basic* and *linear* terms, like $\mathbb{P}((X{=}0 \vee Y{=}1) \wedge (X{=}0 \vee Y{=}0)){=}1 \wedge (\mathbb{P}(X{=}0){=}0 \vee \mathbb{P}(X{=}0){=}1) \wedge (\mathbb{P}(Y{=}0){=}0 \vee \mathbb{P}(Y{=}0){=}1)$, over binary variables X, Y (which can be seen as a result of reduction from the satisfied Boolean formula $(\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b})$). The authors provide a complete axiomatization for the used logic and investigate the complexity of the probabilistic satisfiability problems $\textsc{Sat}^{base}_{prob}$ and $\textsc{Sat}^{lin}_{prob}$, which ask whether there is a joint probability distribution of $X, Y, \ldots$ that satisfies a given Boolean combination of (in)equalities of basic, respectively linear, terms (for formal definitions, see Sec. 2). They show that both satisfiability problems are NP-complete (cf. Fig. 1). Thus, surprisingly, the complexity is no worse than that of propositional logic. Fagin et al. extend then the language to (in)equalities of *polynomial* terms, with the goal of reasoning about conditional probabilities. They prove that there is a PSPACE algorithm, based on Canny's decision procedure [7], for deciding if such a formula is satisfiable but left the exact complexity open. Recently, Mossé, Ibeling, and Icard, [21] have solved this problem, showing that deciding the satisfiability ($\textsc{Sat}^{poly}_{prob}$) is $\exists\mathbb{R}$-complete. In [21], the authors also investigate the satisfiability problems for the higher, more expressive PCH layers – which are not the subject of our paper – and prove an interesting result, that for (in)equalities of polynomial terms both at the interventional and the counterfactual layer the decision problems still remain $\exists\mathbb{R}$-complete.

The languages used in [11, 21] and also in other relevant works as, e.g., [22, 13, 17], can only represent *marginalization* as an expanded sum since they lack a unary summation operator Σ. Thus, for instance, to express the marginal distribution of a random variable Y over a subset of (binary) variables $\{Z_1, \dots, Z_m\}$ as $\sum_{z_1,\dots,z_m} \mathbb{P}(y, z_1, \dots, z_m)$, an encoding without summation requires an extension $\mathbb{P}(y, Z_1{=}0, \dots, Z_m{=}0) + \dots + \mathbb{P}(y, Z_1{=}1, \dots, Z_m{=}1)$ of exponential size. Thus to analyze the complexity aspects of the standard notation of probability theory, one requires an encoding that directly represents marginalization. In a recent paper [31], the authors introduce the class $\mathsf{succ}\text{-}\exists\mathbb{R}$, and show that the satisfiability ($\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$) for the (in)equalities of *polynomial* terms involving probabilities is $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete.

Thus, $\mathsf{succ}\text{-}\exists\mathbb{R}$-completeness seems to be a meaningful yardstick for measuring computational complexity of decision problems. An interesting task would be to investigate problems involving the reals that have been shown to be in $\mathsf{EXPSPACE}$, but not to be $\mathsf{EXPSPACE}$-complete, which are natural candidates for $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete problems.

Contributions and Structure of the Paper

Below we highlight our main contributions, partially summarized also in Fig. 1.

- We provide the characterization of succ-ETR in terms of nondeterministic real RAMs of exponential time respectively (Sec. 3). Moreover, for the classes over the reals in the sequence of inclusions (2), an upward translation result applies, which implies, e.g., $\mathsf{NEXP} \subsetneq \mathsf{succ}\text{-}\exists\mathbb{R}$ unless $\mathsf{NP} = \exists\mathbb{R}$ which is widely disbelieved (Sec. 4).
- We strength slightly the completeness result (marked as (*d*) in Fig. 1) of [31] and prove the problem $\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$ remains $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete even if we disallow the basic terms to contain conditional probabilities (Sec. 5).
- We show that existential second order logic of real numbers is complete for $\mathsf{succ}\text{-}\exists\mathbb{R}$(Sec. 6).
- PSPACE has natural characterizations in terms of ETR; It coincides both with $\exists\mathbb{R}^{\Pi}$ – the class based on ETR enriched with standard product operator, and with $\mathsf{succ}\text{-}\exists\mathbb{R}_{\text{poly}}$, defined in terms of the succinct variant of ETR with polynomially many variables (Sec. 7).
- $\exists\mathbb{R}^{\Sigma}$ – defined similar to $\exists\mathbb{R}^{\Pi}$, but with the addition of a unary summation operator instead – is contained in $\mathsf{PSPACE} = \exists\mathbb{R}^{\Pi}$. We conjecture that this inclusion is strict, as the class is equivalent to $\mathsf{NP}_{\text{real}}^{\mathsf{VNP}_{\mathbb{R}}}$, machine to be an $\mathsf{NP}_{\text{real}}$ model with a $\mathsf{VNP}_{\mathbb{R}}$ oracle, where $\mathsf{VNP}_{\mathbb{R}}$ denotes Valiant's NP over the reals (Sec. 8.1).
- Unlike the languages devoid of the marginalization operator, the crucial small-model property is no longer satisfied. This property says that any satisfiable formula has a model of size bounded polynomially in the input length. Satisfiability with marginalization and with the additional requirement that there is a small model is complete for $\exists\mathbb{R}^{\Sigma}$ at the probabilistic layer (Sec. 8.2).

2 Preliminaries

Complexity Classes Based on the ETR

The problem succ-ETR and the corresponding class $\mathsf{succ}\text{-}\exists\mathbb{R}$ are defined in [31] as follows. succ-ETR is the set of all Boolean circuits C that encode a true sentence φ as in Equation (1) as follows: Assume that C computes a function $\{0,1\}^N \to \{0,1\}^M$. Then φ is a tree consisting of 2^N nodes, each node being labeled with a symbol of $\{\vee, \wedge, \neg, +, \cdot, <, \leq, =\}$, a constant 0 or 1, or a variable $x_0, \dots x_{2^N-1}$. For the node $i \in \{0,1\}^N$, the circuit computes an encoding $C(i)$ of the description of node i, consisting of the label of i, its parent, and its two children. The tree represents a true sentence, if the value at the root node would become true after applying the operator of each node to the value of its children, whereby the value of constants and variables is given in the obvious way. As in the case of $\exists\mathbb{R}$, to succ-$\exists\mathbb{R}$ belong all languages which are polynomial time many-one reducible to succ-ETR.

Besides succ-ETR, [31] introduce more complete problems for succ-$\exists\mathbb{R}$ as intermediate problems in the hardness proof. Of particular importance is the problem Σ_{vi}-ETR that we already discussed in the Introduction. Formally, the problem is defined as an extension of ETR by adding to the signature an additional summation operator $\sum_{x_j=0}^{1}$ with the following semantics[3]: If an arithmetic term is given by a tree with the top gate $\sum_{x_j=0}^{1}$ and $t(x_1,\dots,x_n)$ is the term computed at the child of the top gate, then the new term computes $\sum_{e=0}^{1} t(x_1,\dots,x_{j-1},e,x_{j+1},\dots,x_n)$, that is, we replace the variable x_j by a summation variable e, which then runs from 0 to 1. By nesting the summation operator, we are able to produce a sum with an exponential number of summands. The main reason why the new summation variables are introduced is due to the fact they can be used to *index* the quantified variables x_i used in Formula (1). Similarly as in succ-ETR, sentences of Σ_{vi}-ETR allow the use of exponentially many variables, however, the formulas are given directly and do not require any succinct encoding.

Probabilistic Languages

We always consider discrete distributions in the probabilistic languages studied in this paper. We represent the values of the random variables as $\mathit{Val} = \{0, 1, \dots, c-1\}$ and denote by $X_1, X_2, \dots, X_n$ the random variables used in the input formula. We assume, w.l.o.g., that they all share the same domain Val. A value of X_i is often denoted by x_i or a natural number. In this section, we describe syntax and semantics of the probabilistic languages.

By an *atomic* event, we mean an event of the form $X = x$, where X is a random variable and x is a value in the domain of X. The language $\mathcal{E}$ of propositional formulas over atomic events is the closure of such events under the Boolean operators $\wedge$ and $\neg$: $\mathbf{p} ::= X = x \mid \neg\mathbf{p} \mid \mathbf{p} \wedge \mathbf{p}$. The probability $\mathbb{P}(\delta)$ for formulas $\delta \in \mathcal{E}$ is called *primitive* or *basic term*, from which we build the probabilistic languages. The expressive power and computational complexity of the languages depend on the operations applied to the primitives.Allowing gradually more complex operators, we describe the languages which are the subject of our studies below. We start with the description of the languages $\mathcal{T}^*$ of terms, using the grammars given below.[4]

$$\begin{array}{ll} \mathcal{T}^{base} & \mathbf{t} ::= \mathbb{P}(\delta) \\ \mathcal{T}^{lin} & \mathbf{t} ::= \mathbb{P}(\delta) \mid \mathbf{t}+\mathbf{t} \\ \mathcal{T}^{poly} & \mathbf{t} ::= \mathbb{P}(\delta) \mid \mathbf{t}+\mathbf{t} \mid -\mathbf{t} \mid \mathbf{t}\cdot\mathbf{t} \\ \mathcal{T}^{poly\langle\Sigma\rangle} & \mathbf{t} ::= \mathbb{P}(\delta) \mid \mathbf{t}+\mathbf{t} \mid -\mathbf{t} \mid \mathbf{t}\cdot\mathbf{t} \mid \sum_x \mathbf{t} \end{array}$$

In the summation operator $\sum_x$, we have a dummy variable x which ranges over all values $0, 1, \dots, c-1$. The summation $\sum_x \mathbf{t}$ is a purely syntactical concept which represents the sum $\mathbf{t}[^0/_x] + \mathbf{t}[^1/_x] + \dots + \mathbf{t}[^{c-1}/_x]$, where by $\mathbf{t}[^v/_x]$, we mean the expression in which all occurrences of x are replaced with value v. For example, for $\mathit{Val} = \{0,1\}$, the expression

[3] Recall, in [31], the authors assume arbitrary integer lower and upper bound in $\sum_{x_j=a}^{b}$. But it is easy to see that, w.l.o.g., one can restrict a and b to binary values.

[4] In the given grammars we omit the brackets for readability, but we assume that they can be used in a standard way.

$\sum_x \mathbb{P}(Y{=}1, X{=}x)$ semantically represents $\mathbb{P}(Y{=}1, X{=}0) + \mathbb{P}(Y{=}1, X{=}1)$. We note that the dummy variable x is not a (random) variable in the usual sense and that its scope is defined in the standard way.

The polynomial calculus $\mathcal{T}^{poly}$ was originally introduced by Fagin, Halpern, and Megiddo [11] to be able to express conditional probabilities by clearing denominators. While this works for $\mathcal{T}^{poly}$, this does not work in the case of $\mathcal{T}^{poly\langle\Sigma\rangle}$, since clearing denominators with exponential sums creates expressions that are too large. But we could introduce basic terms of the form $\mathbb{P}(\delta'|\delta)$ with $\delta, \delta' \in \mathcal{E}$ explicitly. All our hardness proofs work without conditional probabilities but all our matching upper bounds are still true with explicit conditional probabilities. For example, expression as $\mathbb{P}(X{=}1) + \mathbb{P}(Y{=}2) \cdot \mathbb{P}(Y{=}3)$ is a valid term in $\mathcal{T}^{poly}$.

Now, let $Lab = \{\text{base}, \text{lin}, \text{poly}, \text{poly}\langle\Sigma\rangle\}$ denote the labels of all variants of languages. Then for each $* \in Lab$ we define the languages $\mathcal{L}^*$ of Boolean combinations of inequalities in a standard way: $\mathbf{f} ::= \mathbf{t} \leq \mathbf{t}' \mid \neg\mathbf{f} \mid \mathbf{f} \wedge \mathbf{f}$, where $\mathbf{t}, \mathbf{t}'$ are terms in $\mathcal{T}^*$.

Although the language and its operations may appear rather restricted, all the usual elements of probabilistic formulas can be encoded. Namely, equality is encoded as greater-or-equal in both directions, e.g. $\mathbb{P}(x) = \mathbb{P}(y)$ means $\mathbb{P}(x) \geq \mathbb{P}(y) \wedge \mathbb{P}(y) \geq \mathbb{P}(x)$. The number 0 can be encoded as an inconsistent probability, i.e., $\mathbb{P}(X{=}1 \wedge X{=}2)$. In a language allowing addition and multiplication, any positive integer can be easily encoded from the fact $\mathbb{P}(\top) \equiv 1$, e.g. $4 \equiv (1{+}1)(1{+}1) \equiv (\mathbb{P}(\top){+}\mathbb{P}(\top))(\mathbb{P}(\top){+}\mathbb{P}(\top))$. If a language does not allow multiplication, one can show that the encoding is still possible. Note that these encodings barely change the size of the expressions, so allowing or disallowing these additional operators does not affect any complexity results involving these expressions.

We define the semantics of the languages as follows. Let $\mathfrak{M} = (\{X_1,\dots,X_n\}, P)$ be a tuple, where P is the joint probability distribution of variables $X_1,\dots,X_n$. For values $x_1,\dots,x_n \in Val$ and $\delta \in \mathcal{E}$, we write $x_1,\dots,x_n \models \delta$ if δ is satisfied by the assignment $X_1{=}x_1,\dots,X_n{=}x_n$. Denote by $S_\delta = \{x_1,\dots,x_n \mid x_1,\dots,x_n \models \delta\}$. We define $[\![\mathbf{e}]\!]_{\mathfrak{M}}$, for some expression $\mathbf{e}$, recursively in a natural way, starting with basic terms as follows $[\![\mathbb{P}(\delta)]\!]_{\mathfrak{M}} = \sum_{x_1,\dots,x_n \in S_\delta} P(X_1{=}x_1,\dots,X_n{=}x_n)$ and $[\![\mathbb{P}(\delta|\delta')]\!]_{\mathfrak{M}} = [\![\mathbb{P}(\delta \wedge \delta')]\!]_{\mathfrak{M}} / [\![\mathbb{P}(\delta')]\!]_{\mathfrak{M}}$, assuming that the expression is undefined if $[\![\mathbb{P}(\delta')]\!]_{\mathfrak{M}} = 0$. For two expressions $\mathbf{e}_1$ and $\mathbf{e}_2$, we define $\mathfrak{M} \models \mathbf{e}_1 \leq \mathbf{e}_2$, if and only if, $[\![\mathbf{e}_1]\!]_{\mathfrak{M}} \leq [\![\mathbf{e}_2]\!]_{\mathfrak{M}}$. The semantics for negation and conjunction are defined in the usual way, giving the semantics for $\mathfrak{M} \models \varphi$ for any $\varphi \in \mathcal{L}^*$.

Existential Second Order Logic of Real Numbers

We follow the definitions of [16]. Let A be a non-empty finite set and $\mathfrak{A} = (A, \mathbb{R}, f_1^{\mathfrak{A}},\dots, f_r^{\mathfrak{A}}, g_1^{\mathfrak{A}},\dots, g_t^{\mathfrak{A}})$, with $f_i^{\mathfrak{A}} : A^{ar(f_i)} \to \mathbb{R}$ and $g_i^{\mathfrak{A}} \subseteq A^{ar(g_i)}$, be a structure. Each $g_i^{\mathfrak{A}}$ is an $ar(g_i)$-ary relation on A and each $f_i^{\mathfrak{A}}$ is a weighted real function on $A^{ar(f_i)}$. The term $\mathbf{t}$ is generated by the following grammar: $\mathbf{t} ::= c \mid f(\vec{x}) \mid \mathbf{t} + \mathbf{t} \mid \mathbf{t} - \mathbf{t} \mid \mathbf{t} \times \mathbf{t} \mid \sum_x \mathbf{t}$, where $c \in \mathbb{R}$ is a constant (denoting itself), f is a function symbol, and $\vec{x}$ is a tuple of first-order variables. An assignment s is a total function that assigns a value in A for each first-order variable. The numerical value of $\mathbf{t}$ in a structure $\mathfrak{A}$ under an assignment s, denoted by $[\![\mathbf{t}]\!]^s_{\mathfrak{A}}$, is defined recursively in a natural way, starting with $[\![f_i(\vec{x})]\!]^s_{\mathfrak{A}} = f_i^{\mathfrak{A}}(s(\vec{x}))$ and applying the standard rules of real arithmetic.

For operators $O \subseteq \{+, \times, \Sigma, -\}$, (in-)equality operators $E \subseteq \{\leq, <, =\}$, and constants $C \subseteq \mathbb{R}$, the grammar of $\mathsf{ESO}_{\mathbb{R}}(O, E, C)$ sentences is given by $\phi ::= x = y \mid \neg(x = y) \mid i\ e\ j \mid \neg(i\ e\ j) \mid R(\vec{x}) \mid \neg R(\vec{x}) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x\ \phi \mid \forall x\ \phi \mid \exists f\ \phi$, where $x, y \in A$ are first order variables, i, j are real terms constructed using operations from O and constants from C, $e \in E$, and R denotes a relation symbol of a finite relational vocabulary[5] $g_1, \dots, g_t$.

[5] The grammar of [16] does not allow quantification over relations, e.g. $\exists R$, as these relations can be replaced by functions, e.g. chosen by $\exists f$.

The semantics of $\mathsf{ESO}_{\mathbb{R}}(O, E, C)$ is defined via $\mathbb{R}$-structures and assignments analogous to first-order logic with additional semantics for second order existential quantifier $\exists f$. That is, a structure $\mathfrak{A}$ satisfies a sentence ϕ under an assignment s, i.e., $\mathfrak{A} \models_s \phi$, according to the following cases of the grammar: $\mathfrak{A} \models_s x = y$, iff $s(x)$ equals $s(y)$; $\mathfrak{A} \models_s \neg(\phi)$ iff $\mathfrak{A} \not\models_s \phi$; $\mathfrak{A} \models_s i\ e\ j$ iff $[\![i]\!]^s_{\mathfrak{A}}\ e\ [\![j]\!]^s_{\mathfrak{A}}$ where $[\![i]\!]^s_{\mathfrak{A}}$ is the numerical value of i as defined above; $\mathfrak{A} \models_s R(\vec{x})$ iff $g_i^{\mathfrak{A}}(s(\vec{x}))$ is true for the $g_i^{\mathfrak{A}}$ corresponding to R in the model $\mathfrak{A}$; $\mathfrak{A} \models_s \phi \wedge \phi'$ iff $\mathfrak{A} \models_s \phi$ and $\mathfrak{A} \models_s \phi'$; $\mathfrak{A} \models_s \phi \vee \phi'$ iff $\mathfrak{A} \models_s \phi$ or $\mathfrak{A} \models_s \phi'$; $\mathfrak{A} \models_s \exists x \phi$ iff $\mathfrak{A} \models_{s[a/x]} \phi$ for some $a \in A$ where $s[a/x]$ means the assignment s modified to assign a to x; $\mathfrak{A} \models_s \forall x \phi$ iff $\mathfrak{A} \models_{s[a/x]} \phi$ for all $a \in A$; and $\mathfrak{A} \models_s \exists f \phi$ iff $\mathfrak{A}[h/f] \models_s \phi$ for some[6] function $h : A^{ar(f)} \to \mathbb{R}$ where $\mathfrak{A}[h/f]$ is the expansion of $\mathfrak{A}$ that interprets f as h.

For a set $S \subseteq \mathbb{R}$, we consider the restricted logic $\mathsf{ESO}_S(O, E, C)$ and $\mathsf{L\text{-}ESO}_S(O, E, C)$. There only the operators and constants of $O \cup E \cup C$ are allowed and all functions f are maps into S, i.e. $f : A^{ar(f)} \to S$. In the loose fragment $\mathsf{L\text{-}ESO}_S(O, E, C)$, negations $\neg(i\ e\ j)$ on real terms are also disallowed.

Probabilistic independence logic $\mathsf{FO}(\perp\!\!\!\perp_c)$ is defined as the extension of first-order logic with probabilistic independence atoms $\vec{x} \perp\!\!\!\perp_{\vec{z}} \vec{y}$ whose semantics is the standard semantics of conditional independence in probability distributions [9, 16].

Known Completeness and Complexity Results

The decision problems $\textsc{Sat}^{*}_{prob}$, with $* \in Lab$, take as input a formula φ in the languages $\mathcal{L}^{*}$ and ask whether there exists a model $\mathfrak{M}$ such that $\mathfrak{M} \models \varphi$. The computational complexity of probabilistic satisfiability problems has been a subject of intensive studies for languages which do not allow explicitly marginalization via summation operator Σ. Very recently [31] addressed the problem for polynomial languages.

Below, we summarize these results[7], informally presented in the Introduction:

- $\textsc{Sat}^{base}_{prob}$ and $\textsc{Sat}^{lin}_{prob}$ are NP-complete, [11],
- $\textsc{Sat}^{poly}_{prob}$ is $\exists\mathbb{R}$-complete [21], and
- $\textsc{Sat}^{poly\langle\Sigma\rangle}_{prob}$ is $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete [31].

For a logic L, the satisfiability problem $\textsc{Sat}(L)$ is defined as follows: given a formula $\varphi \in L$, decide whether φ is satisfiable. For the model checking problem of a logic L, we consider the following variant: given a sentence $\varphi \in L$ and a structure $\mathfrak{A}$, decide whether $\mathfrak{A} \models \varphi$. For model checking of $\mathsf{FO}(\perp\!\!\!\perp_c)$, the best-known complexity lower and upper bounds are NEXP-hardness and $\mathsf{EXPSPACE}$, respectively [15].

3 NEXP over the Reals

In [10], Erickson, van Der Hoog, and Miltzow extend the definition of word RAMs to real computations. In contrast to the so-called BSS model of real computation [5], the real RAMs of Erickson et al. provide integer and real computations at the same time, allowing for instance indirect memory access to the real registers and other features that are important to implement algorithms over the reals. The input to a real RAM is a pair of vectors, the first one is a vector of real numbers, the second is a vector of integers. Real RAMs have two types of registers, word registers and real registers. The word registers can store integers

[6] Note that h might be an arbitrary function and is not restricted to the functions $f_i^{\mathfrak{A}}$ of the model.

[7] In the papers [21] and [31] the authors show even stronger results, namely that the completeness results also hold for causal satisfiability problems.

with w bits, where w is the word size. The total number of registers is 2^w for each of the two types. Real RAMs perform arithmetic operations on the word registers, where words are interpreted as integers between 0 and $2^w - 1$, and bitwise Boolean operations. On the real registers, only arithmetic operations are allowed. Word registers can be used for indirect addressing on both types of registers and the control flow is established by conditional jumps that depend on the result of a comparison of two word registers or of a real register with the constant 0. For further details we refer to the original paper [10].

The real RAMs of [10] characterize the existential theory of the reals. The authors prove that a problem is in $\exists\mathbb{R}$ iff there is a polynomial time real verification algorithm for it. In this way, real RAMs are an “easy to program” mechanism to prove that a problem is contained in $\exists\mathbb{R}$. Beside the input I, which is a sequence of words, the real verification algorithm also gets a certificate consisting of a sequence of real numbers x and a further sequence of words z. I is in the language if there is a pair (x, z) that makes the real verification algorithm accept. I is not in the language if for all pairs (x, z), the real verification rejects.

Instead of using certificates and verifiers, we can also define nondeterministic real RAMs that can guess words and real numbers on the fly. Like for classical Turing machines, it is easy to see that these two definitions are equivalent (when dealing with time bounded computations).

▶ **Definition 1.** *Let $t : \mathbb{N} \to \mathbb{N}$ be a function. We define $\mathsf{NTime}_{real}(t)$ to be the set of all languages $L \subseteq \{0,1\}^\star$, such that there is a constant $c \in \mathbb{N}$ and a nondeterministic real word-RAM M that recognizes L in time t for all word-sizes $w \geq c \cdot \log(t(n)) + c$.*

For any set of functions T, we define $\mathsf{NTime}_{real}(T) = \bigcup_{t\in T} \mathsf{NTime}_{real}(t)$. We define our two main classes of interest, NP_{real} and NEXP_{real} as follows:

$$\mathsf{NP}_{real} = \mathsf{NTime}_{real}(\text{poly}(n)), \qquad \mathsf{NEXP}_{real} = \mathsf{NTime}_{real}(2^{\text{poly}(n)}).$$

Note that the word size needs to be at least logarithmic in the running time, to be able to address a new register in each step.

One of the main results of Erickson et al. (Theorem 2 in their paper) can be rephrased as $\exists\mathbb{R} = \mathsf{NP}_{\text{real}}$. Their techniques can be extended to prove that $\mathsf{succ}\text{-}\exists\mathbb{R} = \mathsf{NEXP}_{\text{real}}$.

We get the following in analogy to the well-known results that the succinct version of 3-Sat is NEXP-complete.

▶ **Lemma 2.** succ-*ETR is* NEXP_{real}*-complete and thus* $\mathsf{NEXP}_{real} = \mathsf{succ}\text{-}\exists\mathbb{R}$.

Proof idea. For the one direction, one carefully has to analyze the construction by Erickson et al. and show that the simulation there can also be implemented succinctly. The reverse direction simply follows from expanding the succinct ETR instance and use the fact that nondeterministic real word-RAMs can solve ETR in polynomial time. Along the way, we also obtain a useful normalization procedure for succinct ETR instances. While for normal ETR instances, it is obvious that one can always push negations down, it is not clear for succinct instances. We describe the details in the full version. ◀

4 The Relationships between the Boolean Classes and Classes over the Reals

Now we study the new class $\mathsf{NEXP}_{\text{real}} = \mathsf{succ}\text{-}\exists\mathbb{R}$ from a complexity theoretic point of view.

$$\mathsf{NP} \subseteq \exists\mathbb{R} \subseteq \mathsf{PSPACE}; \qquad \mathsf{NEXP} \subseteq \mathsf{succ}\text{-}\exists\mathbb{R} \subseteq \mathsf{EXPSPACE}. \tag{3}$$

The left side of (3) is well-known. The first inclusion on the right side is obvious, since a real RAM simply can ignore the real part. The second inclusion follows from expanding the succinct instance into an explicit formula (which now has exponential size) and simply using the known PSPACE-algorithm.

We prove two translation results, that is, equality of one of the inclusion in the left equation of (3) implies the equality of the corresponding inclusion in the right equation of (3).

▶ **Theorem 3.** *If* $\exists\mathbb{R} = \mathsf{NP}$*, then* $\mathsf{succ}\text{-}\exists\mathbb{R} = \mathsf{NEXP}$.

▶ **Theorem 4.** *If* $\exists\mathbb{R} = \mathsf{PSPACE}$*, then* $\mathsf{succ}\text{-}\exists\mathbb{R} = \mathsf{EXPSPACE}$.

Further we prove a nondeterministic time hierarchy theorem (see the full version for the details) for real word RAMs. Using the characterization of $\exists\mathbb{R}$ and $\mathsf{succ}\text{-}\exists\mathbb{R}$ in terms of real word RAMs, in particular, we get that succ-ETR is strictly more expressive than ETR.

▶ **Corollary 5.** $\exists\mathbb{R} = \mathsf{NP}_{real} \subsetneq \mathsf{NEXP}_{real} = \mathsf{succ}\text{-}\exists\mathbb{R}$.

5 Hardness of Probabilistic Satisfiability without Conditioning

To prove that $\text{SAT}_{prob}^{poly\langle\Sigma\rangle}$ is $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete, van der Zander, Bläser and Liśkiewicz [31] show the hardness part for the variant of the probabilistic language where the primitives are also allowed to be conditional probabilities. A novel contribution of our work is to extend this completeness result to our version for languages which disallow conditional probabilities:

▶ **Theorem 6.** *The problem* $\text{SAT}_{prob}^{poly\langle\Sigma\rangle}$ *remains* $\mathsf{succ}\text{-}\exists\mathbb{R}$*-complete even without conditional probabilities.*

In the rest of this section, we will give the proof of the theorem.

In [31] the authors have already shown that Σ_{vi}-ETR is $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete. We define Σ_{vi}-ETR$_1$ in the same way as Σ_{vi}-ETR, but asking the question whether there is a solution where the sum of the absolute values (ℓ_1 norm) is bounded by 1. Then we can reduce Σ_{vi}-ETR$_1$ to $\text{SAT}_{prob}^{poly\langle\Sigma\rangle}$ without the need for conditional probabilities (Lemma 9). The proof that Σ_{vi}-ETR$_1$ is hard for $\mathsf{succ}\text{-}\exists\mathbb{R}$ (Lemma 8) depends on a result of Grigoriev and Vorobjov [14] who showed that the solution to an ETR instance can be bounded by a constant that only depends on the bitsize of the instance. Thus the solution can be scaled to fit into a probability distribution. This completes the proof of Theorem 6.

▶ **Theorem 7** (Grigoriev and Vorobjov [14])**.** *Let* $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ *be polynomials of total degree* $\leq d$ *with coefficients of bit size* $\leq L$*. Then every connected component of* $\{x \in \mathbb{R}^n \mid f_1(x) \geq 0 \wedge \dots \wedge f_k(x) \geq 0\}$ *contains a point of distance less than* $2^{Ld^{cn}}$ *from the origin for some absolute constant* c*. The same is true if some of the inequalities are replaced by strict inequalities.*

▶ **Lemma 8.** Σ_{vi}*-ETR* $\leq_P \Sigma_{vi}$*-ETR*$_1$*.*

Proof. Let ϕ be an instance of Σ_{vi}-ETR. We will transform it into a formula φ such that φ has a solution with ℓ_1 norm bounded by 1 iff ϕ has any solution.

Let S be the bit length of ϕ. The number n of variables in ϕ is bounded by 2^S. The degree of all polynomials is bounded by S. Note that the exponential sums do not increase the degree at all. Finally, all coefficients have bit size $O(S)$. Note that one summation operator doubles the coefficients at most. By Theorem 7, if ϕ is satisfiable, then there is a solution with entries bounded by $T := 2^{2^{2^{cS}}}$ for some constant c.

In our new instance φ first creates a small constant $d \leq 1/((2^m + n)T)$ for some m polynomial in S defined below. This can be done using Tseitin's trick: We take 2^m many fresh variables t_i and start with $(2^m + 2^S)t_1 = 1$ and then iterate by adding the equation $\sum_{i=1}^{2^m-1}(t_i^2 - t_{i+1})^2 = 0$, i.e. forcing $t_{i+1} = t_i^2$. To implement the first equation we replace 2^m by $\sum_{e_1=0}^{1} \cdots \sum_{e_m=0}^{1} 1$ and similarly replace 2^S. To implement the second equation we replace $\sum_{i=1}^{2^m-1}(t_i^2 - t_{i+1})^2$ by $\sum_{e_1=0}^{1} \cdots \sum_{e_m=0}^{1} \sum_{f_1=0}^{1} \cdots \sum_{f_m=0}^{1} (t_{e_1,\ldots,e_m}^2 - t_{f_1,\ldots,f_m})^2 \cdot A(e_1, \ldots, e_m, f_1, \ldots, f_m)$ where A is an arithmetic formula returning 1 iff the binary number represented by $f_1, \ldots, f_m$ is the successor of the binary number represented by $e_1, \ldots, e_m$. The number m is polynomial in S. The unique satisfying assignment to the t_i has its entries bounded by $1/(2^m + 2^S)$. Let $d := t_{2^m}$ be the last variable.

Now in ϕ we replace every occurrence of x_i by ${}^{x_i}/_{d}$ and then multiple each (in-)equality by an appropriate power of d to remove the divisions in order to obtain φ. In this way, from every solution to ϕ, we obtain a solution to φ by multiplying the entries by d and vice versa. Whenever ϕ has a solution, then it has one with entries bounded by T. By construction φ then has a solution with entries bounded by $1/(2^m + 2^S)$. Since each entry of the solution is bounded by $1/(2^m + 2^S)$, the ℓ_1 norm is bounded by $1/2$. ◀

▶ **Lemma 9.** *Σ_{vi}-ETR$_1$ $\leq_P$ $\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$ via a reduction without the need for conditional probabilities.*

Proof. Let X_0 be a random variable with range $\{-1, 0, 1\}$ and let $X_1, \ldots, X_N$ be binary random variables. We replace each real variable $x_{e_1,\ldots,e_N}$ in the Σ_{vi}-ETR$_1$ formula as follows:

$$x_{e_1,\ldots,e_N} := \mathbb{P}(X_0{=}1 \wedge X_1{=}e_1 \wedge \ldots \wedge X_N{=}e_N) - \mathbb{P}(X_0{=}-1 \wedge X_1{=}e_1 \wedge \ldots \wedge X_N{=}e_N)$$

This guarantees that $x_{e_1,\ldots,e_N} \in [-1, 1]$. The existential quantifiers now directly correspond to the existence of a probability distribution $P(X_0, \ldots, X_N)$, where each variable corresponds to an different set of two entries of P.

Let $P(X_0, \ldots, X_N)$ be a solution to the constructed $\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$ instance. Then clearly setting $x_{e_1,\ldots,e_N} = P(1, e_1, \ldots, e_N) - P(-1, e_1, \ldots, e_N)$ satisfies the original Σ_{vi}-ETR$_1$ instance. Furthermore it has an ℓ_1 norm bounded by 1:

$$\begin{aligned}\sum_{e_1=0}^{1} \cdots \sum_{e_N=0}^{1} |x_{e_1,\ldots,e_N}| &= \sum_{e_1=0}^{1} \cdots \sum_{e_N=0}^{1} |P(1, e_1, \ldots, e_N) - P(-1, e_1, \ldots, e_N)| \\ &\leq \sum_{e_1=0}^{1} \cdots \sum_{e_N=0}^{1} (P(1, e_1, \ldots, e_N) + P(-1, e_1, \ldots, e_N)) \\ &\leq 1\,.\end{aligned}$$

Vice-versa, let the original Σ_{vi}-ETR$_1$ be satisfied by some choice of the $x_{e_1,\ldots,e_N}$ with ℓ_1 norm α bounded by 1. We define the probability distribution

$$P(X_0, X_1, \ldots, X_N) = \begin{cases} \frac{1-\alpha}{2^N} & \text{if } X_0 = 0 \\ \max(x_{X_1,\ldots,X_N}, 0) & \text{if } X_0 = 1 \\ \max(-x_{X_1,\ldots,X_N}, 0) & \text{if } X_0 = -1 \end{cases}$$

Every entry of P is non-negative since $\alpha \leq 1$. Furthermore the sum of all entries is exactly 1, the entries with $X_0 \in \{-1, 1\}$ contribute exactly α total and the 2^N entries with $X_0 = 0$ contribute $1 - \alpha$ total. Since P fulfills the equation $x_{e_1,\ldots,e_N} = P(1, e_1, \ldots, e_N) - P(-1, e_1, \ldots, e_N)$, it is a solution to the constructed $\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$ instance. ◀

6 Correspondence to Existential Second Order Logic and $\mathsf{FO}(\perp\!\!\!\perp_c)$

In this section, we investigate the complexity of existential second order logics and the probabilistic independence logic $\mathsf{FO}(\perp\!\!\!\perp_c)$.

▶ **Lemma 10.** *Model checking of* $\mathsf{ESO}_{\mathbb{R}}(\Sigma, +, \times, \leq, <, =, \mathbb{Q})$ *is in* $\mathsf{succ}\text{-}\exists\mathbb{R}$*.*

Proof. In model checking, the input is a finite structure $\mathfrak{A}$ and a sentence ϕ, and we need to decide whether $\mathfrak{A} \models \phi$. $\mathfrak{A}$ includes a domain A for the existential/universal quantifiers over variables. Any function (relation) of arity k can be represented as a (Boolean) table of size $|A|^k$. Some of these tables might be given in the input. The remaining tables of functions chosen by quantifiers $\exists f$ can simply be guessed by a $\mathsf{NEXP}_{\text{real}}$ machine in non-deterministic exponential time. Then all possible values for the quantifiers of the finite domain can be enumerated and all sentences can be evaluated. This completes the proof, due to the characterization given in Lemma 2. ◀

▶ **Proposition 11.** *Model checking of* $\mathsf{L\text{-}ESO}_{[0,1]}(+, \times, \leq, 0, 1)$ *is* $\mathsf{succ}\text{-}\exists\mathbb{R}$*-hard.*

Proof. We start with the following equivalences relating the logics:

$$\mathsf{L\text{-}ESO}_{[0,1]}(+, \times, \leq, 0, 1) \equiv \mathsf{L\text{-}ESO}_{[-1,1]}(+, \times, \leq, 0, 1) \equiv \mathsf{L\text{-}ESO}_{[-1,1]}(+, \times, -, =, \leq, 0, {}^1\!/\!{}_8, 1).$$

The first equivalence has been shown by Hannula et al. [16]. To see the second one, note that we can replace operator $=$ using $a = b$ as $a \leq b \wedge b \leq a$. The negative one -1 can be defined by a function -1 of arity 0 using $\exists(-1) : (-1) + 1 = 0$. Then any subtraction $a - b$ can be replaced with $a + (-1) \times b$. Finally, the fraction ${}^1\!/\!{}_8$ is a function given by $\exists {}^1\!/\!{}_8 : {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 + {}^1\!/\!{}_8 = 1$. These equivalence reductions can be performed in polynomial time.

In the rest of the proof, we show the hardness, reducing the problems in $\mathsf{succ}\text{-}\exists\mathbb{R}$ to the existential second order logic $\mathsf{L\text{-}ESO}_{[-1,1]}(+, \times, -, =, \leq, 0, {}^1\!/\!{}_8, 1)$. To this aim, we use a $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete problem which is based on a problem given by Abrahamsen et al. [1], who have shown that an equation system consisting of only sentences of the form $x_i = {}^1\!/\!{}_8$, $x_i + x_j = x_k$, and $x_i \cdot x_j = x_k$ is $\exists\mathbb{R}$-complete. As shown in [31], this can be turned into a $\mathsf{succ}\text{-}\exists\mathbb{R}$-complete problem, denoted as $\mathrm{succETR}^{{}^1\!/\!{}_8,+,\times}_{[-{}^1\!/\!{}_8,{}^1\!/\!{}_8]}$, by replacing the explicit indices i, j, k with circuits that compute the indices for an exponential number of these three equations. The circuits can be encoded with arithmetic operators, which allows us to encode all equations in existential second order logic in a polynomial time reduction.

The instances of $\mathrm{succETR}^{{}^1\!/\!{}_8,+,\times}_{[-{}^1\!/\!{}_8,{}^1\!/\!{}_8]}$ are represented as seven Boolean circuits $C_0, C_1, \ldots, C_6 : \{0,1\}^M \to \{0,1\}^N$ such that $C_0(j)$ gives the index of the variable in the jth equation of type $x_i = {}^1\!/\!{}_8$, $C_1(j), C_2(j), C_3(j)$ give the indices of variables in the jth equation of the type $x_{i_1} + x_{i_2} = x_{i_3}$, and $C_4(j), C_5(j), C_6(j)$ give the indices of variables in the jth equation of the type $x_{i_1} x_{i_2} = x_{i_3}$. Without loss of generality, we can assume that each type has the same number 2^M of equations. An instance of the problem $\mathrm{succETR}^{{}^1\!/\!{}_8,+,\times}_{[-{}^1\!/\!{}_8,{}^1\!/\!{}_8]}$ is satisfiable if and only if:

$$\begin{aligned}&\exists x_0, \ldots, x_{2^N-1} \in [-{}^1\!/\!{}_8, {}^1\!/\!{}_8] : \forall j \in [0, 2^M - 1] :\\ &\qquad x_{C_0(j)} = {}^1\!/\!{}_8, \quad x_{C_1(j)} + x_{C_2(j)} = x_{C_3(j)}, \quad \text{and} \quad x_{C_4(j)} \cdot x_{C_5(j)} = x_{C_6(j)}. \end{aligned} \tag{4}$$

Below, we prove that

$$\mathrm{succETR}^{{}^1\!/\!{}_8,+,\times}_{[-{}^1\!/\!{}_8,{}^1\!/\!{}_8]} \leq_P \mathsf{L\text{-}ESO}_{[-1,1]}(+, \times, -, =, \leq, 0, {}^1\!/\!{}_8, 1). \tag{5}$$

Let the instance of $\mathrm{succETR}^{1/8,+,\times}_{[-1/8,1/8]}$ be represented by seven Boolean circuits $C_0, C_1,\ldots, C_6 : \{0,1\}^M \to \{0,1\}^N$ as described above. Let the variables of the instance be indexed as $x_{e_1,\ldots,e_N}$, with $e_i \in \{0,1\}$ for $i \in [N]$. We will identify the bit sequence $\vec{b} = b_1,\ldots, b_M$ by an integer j, with $0 \le j \le 2^M - 1$, the binary representation of which is $b_1\ldots b_M$ and vice versa.

We construct sentences in the logic $\mathsf{L\text{-}ESO}_{[-1,1]}(+, \times, -, =, \le, 0, 1/8, 1)$ and prove that a binary model satisfies the sentences if and only if the formula (4) is satisfiable.

Let q be an N-ary function where $q(e_1, \ldots, e_N)$ should encode the value of variable $x_{e_1,\ldots,e_N}$. For the range, we require $\forall \vec{x} : 0 - 1/8 \le q(\vec{x}) \wedge q(\vec{x}) \le 1/8$.

For each circuit C_i, we define a function y_i whose value $y_i(\vec{b})$ is $x_{C_i(j)}$, i.e., $q(C_i(j))$. Then y_i can directly be inserted in the equation system (4). For this, we need to encode the circuit as logical sentences and relate y and q.

To model a Boolean formula encoded by a node of C_i, with $i = 0, 1,\ldots, 6$, we use one step of arithmetization to go from logical formulas to calculations on real numbers, where $0 \in \mathbb{R}$ means false and $1 \in \mathbb{R}$ means true. While negation is not allowed directly in L-ESO, on the real numbers we can simulate negation by subtraction.

For each node v of each circuit C_i, we need a function $c_{i,v}$ of arity M, such that $c_{i,v}(\vec{b})$ is the value computed by the node if the circuit is evaluated on input $j = b_1\ldots b_M$.

If v is an input node, the node only reads one bit $u_{i,k}$ from the input, so let $\forall \vec{b} : c_{i,v}(\vec{b}) = id(b_k)$, where id is a function that maps $0, 1$ from the finite domain to $0, 1 \in \mathbb{R}$.

For each internal node v of C_i, we proceed as follows.

If v is labeled with $\neg$ and u is a child of v, then we require $\forall \vec{b} : c_{i,v}(\vec{b}) = 1 - c_{i,u}(\vec{b})$.

If v is labeled with $\wedge$ and u and w are children of v, then we require $\forall \vec{b} : c_{i,v}(\vec{b}) = c_{i,u}(\vec{b}) \times c_{i,w}(\vec{b})$.

Finally, if v is labeled with $\vee$ and u and w are children of v, then we require $\forall \vec{b} : c_{i,v}(\vec{b}) = 1 - (1 - c_{i,u}(\vec{b})) \times (1 - c_{i,w}(\vec{b}))$.

Thus, if v is an output node of a circuit C_i, then, for C_i fed with input $j = b_1\ldots b_M \in \{0,1\}^M$, we have that v evaluates to true if and only if $c_{i,v}(\vec{b}) = 1$.

Next, we need an $(N + M)$-arity selector function $s_i(\vec{b}, \vec{e})$ which returns 1 iff the output of circuit C_i on input $\vec{b}$ is $\vec{e}$. It can be defined as:

$$\forall \vec{b}, \vec{e} : \quad s_i(\vec{b}, \vec{e}) = \prod_{k=1}^{N} (c_{i,v_k}(\vec{b}) \times id(e_k) + (1 - c_{i,v_k}(\vec{b})) \times (1 - id(e_k))).$$

Each factor of the product is 1 iff $c_{i,v_k}(\vec{b}) = e_k$. It has constant length, so it can be expanded using the multiplication of the logic.

We express each $q(C_i(j))$ as a function $y_i(j)$, where $\vec{b}$ is the binary representation of j:

$$\forall \vec{b}, \vec{e} : \quad y_i(\vec{b}) \times s_i(\vec{b}, \vec{e}) = q(\vec{e}) \times s_i(\vec{b}, \vec{e}).$$

The above equation is trivially satisfied for $s_i(\vec{b}, \vec{e}) = 0$, thus it enforces equality of $y_i(\vec{b})$ and $q(\vec{e})$ only in the case $s_i(\vec{b}, \vec{e}) = 1$. Inserting y_i in the equation system (4) gives us the last L-ESO formula:

$$\forall \vec{b} : \quad y_0(\vec{b}) = 1/8, \quad y_1(\vec{b}) + y_2(\vec{b}) = y_3(\vec{b}), \quad \text{and} \quad y_4(\vec{b}) \times y_5(\vec{b}) = y_6(\vec{b}),$$

which, combining with the previous formulas and preceded by second order existential quantifiers $\exists y_i, \exists s_i, \exists c_{i,v}, \exists id$, with $i = 0, \ldots, 6$, is satisfiable if and only if the formula (4) are satisfiable.

Obviously, the size of the resulting sentences are polynomial in the size $|C_0|+|C_1|+\ldots+|C_6|$ of the input instance and the sentences can be computed in polynomial time.

This completes the construction of reduction (5) and the proof of the proposition. ◀

As $\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,\leq,0,1)$ is weaker than $\mathsf{ESO}_{\mathbb{R}}(\Sigma,+,\times,\leq,<,=,\mathbb{Q})$, it follows:

▶ **Theorem 12.** *Let $S=\mathbb{R}$ or $S=[a,b]$ with $[0,1]\subseteq S$, $\{0,1\}\subseteq C\subseteq\mathbb{Q}$, $\{\times\}\subseteq O\subseteq\{+,\times,\Sigma\}$ with $|O|\geq 2$,and $E\subseteq\{\leq,<,=\}$ with $\{\leq,=\}\cap E\neq\emptyset$. Model checking of*

- $\mathsf{L\text{-}ESO}_S(O,E,C)$ *and*
- $\mathsf{ESO}_S(O,E,C)$

is $\mathsf{succ}\text{-}\exists\mathbb{R}$*-complete.*

Proof. We start with the following equivalence, which follows from the fact that a comparison $a\leq b$ can be replaced by $\exists\epsilon,x:a\epsilon+x=b\epsilon$:

$$\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,\leq,0,1)\equiv\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,=,0,1). \tag{6}$$

The next fact has been used by [16], but without proof. Perhaps the authors thought it to be too trivial to mention. But it is not obvious, since the standard technique of replacing $a\leq b$ with $\exists x:a+x^2=b$ does not work here when x is restricted to $[0,1]$.

In some sense, $\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,\leq,0,1)$ is the weakest logic one can consider in this context:

▶ **Fact 13.** *Let $S=\mathbb{R}$ or $S=[a,b]$ with $[0,1]\subseteq S$, $\{0,1\}\subseteq C\subseteq\mathbb{Q}$, $\{\times\}\subseteq O\subseteq\{+,\times,\Sigma\}$ with $|O|\geq 2$,and $E\in\{\leq,=\}$.*

$\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,\leq,0,1)\leq\mathsf{L\text{-}ESO}_S(O,E,C)\leq\mathsf{ESO}_S(O,E,C)$.

Proof of Fact 13. Relation $=$ subsumes $\leq$ due to (6).

If $+\in O$, the remaining statements are trivial. Otherwise, we need to express $+$ using Σ.

If $S=\mathbb{R}$, $x+y$ can be written as $\Sigma_t c(t)$ where $c(0)=x, c(1)=y$. (we consider model checking problems, where the finite domain can be set to binary)

If $S=[a,b]$, x or y might be outside the range. But the total weight of any k-arity function is $(b-a)^k$ and each term has a maximal polynomial degree D, so x and y are bounded by $O((b-a)^{kD})$. So all expressions can be scaled to fit in the range (Lemma 6.4. Step 3 proves this for functions that are probability distributions in [16]). ◀

All of this combined shows the theorem. ◀

Hannula et. al [16] and Durand et. al [9] have shown the following relationships between expressivity of the logics: $\mathsf{L\text{-}ESO}_{[0,1]}(+,\times,=,0,1)\leq\mathsf{L\text{-}ESO}_{d[0,1]}(\Sigma,\times,=)\equiv\mathsf{FO}(\perp\!\!\!\perp_c)$. $\mathsf{L\text{-}ESO}_{d[0,1]}[O,E,C]$ means a variant of $\mathsf{L\text{-}ESO}_{[0,1]}(O,E,C)$ where all functions are required to be distributions, that is $f^{\mathfrak{A}}:A^{ar(f)}\to[0,1]$ and $\sum_{\vec{a}\in A^{ar(f)}}f^{\mathfrak{A}}(\vec{a})=1$. From the proof for the translation from $\mathsf{L\text{-}ESO}$ to $\mathsf{FO}(\perp\!\!\!\perp_c)$ in [9], it follows that the reduction can be done in polynomial time. Moreover, it is easy to see that model checking of $\mathsf{FO}(\perp\!\!\!\perp_c)$ can be done in $\mathsf{NEXP}_{\mathrm{real}}$. Thus we get

▶ **Corollary 14.** *Model checking of $\mathsf{FO}(\perp\!\!\!\perp_c)$ is* $\mathsf{succ}\text{-}\exists\mathbb{R}$*-complete.*

This corollary answers the question asked in [15] for the exact complexity of $\mathsf{FO}(\perp\!\!\!\perp_c)$ and confirms their result that the complexity lies between NEXP and $\mathsf{EXPSPACE}$.

7 Succinct ETR of Polynomially Many Variables

The key feature that makes the language Σ_{vi}-ETR defined in [31] very powerful is the ability to index the quantified variables in the scope of summation. Nesting the summations allows handling an exponential number of variables. Thus, similarly as in succ-ETR, sentences of Σ_{vi}-ETR allow the use of exponentially many variables, however, the formulas are given directly and do not require any succinct encoding. Due to the fact that variable indexing is possible, [31] show that Σ_{vi}-ETR is polynomial time equivalent to succ-ETR.

Valiant's class VNP [6, 19] is also defined in terms of exponential sums (we recall the definition of VNP and related concepts in the full version). However, we cannot index variables as above, therefore, the overall number of variables is always bounded by the length of the defining expression. It is natural to extend ETR with a summation operator, but without variable indexing as was allowed in Σ_{vi}-ETR. In this way, we can have exponential sums, but the number of variables is bounded by the length of the formula. Instead of a summation operator, we can also add a product operator, or both.

▶ **Definition 15.**

1. Σ-*ETR is defined as ETR with the addition of a unary summation operator* $\sum_{x_i=0}^{1}$.
2. Π-*ETR is defined similar to* Σ-*ETR, but with the addition of a unary product operator* $\prod_{x_i=0}^{1}$ *instead.*
3. $\Sigma\Pi$-*ETR is defined similar to* Σ-*ETR or* Π-*ETR, but including both unary summation and product operators.*

In the three problems above, the number of variables is naturally bounded by the length of the instance, since the problems are not succinct. For example, the formula $\sum_{x_1=0}^{1}\sum_{x_2=0}^{1}(x_1+x_2)(x_1+(1-x_2))(1-x_1)=0$ explained in the introduction is also in Σ-ETR and $\Sigma\Pi$-ETR, but not in Π-ETR. The formula $\sum_{e_1=0}^{1}\cdots\sum_{e_N=0}^{1}(x_{\langle e_1,\ldots,e_N\rangle})^2=1$ is in neither of these three classes since it uses variable indexing.

To demonstrate the expressiveness of Π-ETR, we will show that the PSPACE-complete problem QBF can be reduced to it.

▶ **Lemma 16.** QBF $\leq_P$ Π-*ETR.*

Proof. Let $Q_1x_1Q_2x_2\ldots Q_nx_n\varphi(x_1,\ldots,x_n)$ be a quantified Boolean formula with $Q_1,\ldots,Q_n\in\{\exists,\forall\}$. We arithmetize φ as $A(\varphi)$ inductively using the following rules:

φ is a variable x_i: We construct $A(\varphi)=x_i$.
φ is $\neg\varphi_1$: We construct $A(\varphi)$ as $1-A(\varphi_1)$.
φ is $\varphi_1\wedge\varphi_2$: We construct $A(\varphi)$ as $A(\varphi_1)\cdot A(\varphi_2)$.
φ is $\varphi_1\vee\varphi_2$: We construct $A(\varphi)$ as $1-(1-A(\varphi_1))\cdot(1-A(\varphi_2))$ via De Morgan's law and the previous two cases.

The special treatment of the $\vee$ operator ensures that whenever $x_1,\ldots,x_n\in\{0,1\}$, then $A(\varphi)$ evaluates to 1 iff $x_1,\ldots,x_n$ satisfy φ and 0 otherwise. We then arithmetize the quantifiers $Q_1,\ldots,Q_n$ in a similar way, but using the unary product operator.

$Q_i=\forall$: We construct $A(\forall x_iQ_{i+1}x_{i+1}\ldots Q_nx_n\varphi)$ as $\prod_{x_i=0}^{1}A(Q_{i+1}x_{i+1}\ldots Q_nx_n\varphi)$
$Q_i=\exists$: We construct $A(\exists x_iQ_{i+1}x_{i+1}\ldots Q_nx_n\varphi(x_1,\ldots,x_n)$ as $1-\prod_{x_i=0}^{1}(1-A(Q_{i+1}x_{i+1}\ldots Q_nx_n\varphi))$, again using De Morgan's law.

The final Π-ETR formula is then just $A(Q_1x_1Q_2x_2\ldots Q_nx_n\varphi)=1$.

The correctness of the construction follows because a formula of the form $\forall\psi(x)$ is true over the Boolean domain iff $\psi(0)\wedge\psi(1)$ is true. The unary product together with arithmetization allows us to write the whole formula down without an exponential blow-up. ◀

We also consider the succinct version of ETR with only a polynomial number of variables.

▶ **Definition 17.** succ-*ETR*$_{\text{poly}}$ *is defined similar to* succ-*ETR, but variables are encoded in unary instead of binary, thus limiting the amount of variables to a polynomial amount of variables. (Note that the given input circuit succinctly encodes an ETR formula and not an arbitrary circuit.)*

For succ-ETR, it does not matter whether the underlying structure of the given instance is a formula or an arbitrary circuit, since we can transform the circuit into a formula using Tseitin's trick. This, however, requires a number of new variables that is proportional to the size of the circuit, which is exponential.

Like for ETR, we can now define corresponding classes by taking the closure of the problems defined above. It turns out that we get meaningful classes in this way, however, for some unexpected reason. Except for Σ-ETR, all classes coincide with PSPACE, which we will see below. By restricting the number of variables to be polynomial, the complexity of succ-ETR reduces considerably, from being NEXP$_{\text{real}}$-complete, which contains NEXP, to PSPACE. On the other hand, the problems are most likely more powerful than ETR, assuming that $\exists\mathbb{R}$ is a proper subset of PSPACE, which is believed by at least some researchers.

▶ **Definition 18.** *Let* succ-$\exists\mathbb{R}_{\text{poly}}$ *be the closure of* succ-*ETR*$_{\text{poly}}$ *under polynomial time many one reductions.*

▶ **Theorem 19.** PSPACE $=$ succ-$\exists\mathbb{R}_{\text{poly}}$ *and the problems* Π-*ETR,* ΣΠ-*ETR, and* succ-*ETR*$_{\text{poly}}$ *are* PSPACE*-complete.*

Proof idea. To show that succ-ETR$_{\text{poly}}$ is in PSPACE, we rely on results by [24]. One of the famous consequences of Renegar's work is that ETR $\in$ PSPACE. But Renegar shows even more, because he can handle an exponential number of arithmetic terms of exponential size with exponential degree as long as the number of variables is polynomially bounded. For the completeness of Π-ETR, it turns out that an unbounded product is able to simulate an arbitrary number of Boolean quantifier alternations, in constrast to an unbounded sum. So, as shown in Lemma 16, we can reduce QBF to it. ◀

8 ETR with the Standard Summation Operator

In the previous section, we have seen that ETR with a unary product operator (Π-ETR) is PSPACE-complete. Moreover, allowing both unary summation and product operators does not lead to an increase in complexity. In this section, we investigate the complexity of Σ-ETR, ETR with only unary summation operators.

▶ **Definition 20.** *Let* $\exists\mathbb{R}^{\Sigma}$ *be the closure of* Σ-*ETR under polynomial time many one reductions. Moreover, for completeness, let* $\exists\mathbb{R}^{\Pi}$ *be the closure of* Π-*ETR under polynomial time many one reductions.*

8.1 Machine Characterization of $\exists\mathbb{R}^{\Sigma}$

By Theorem 19, we have $\exists\mathbb{R}^{\Pi}$ = PSPACE. For $\exists\mathbb{R}^{\Sigma}$, we can conclude: NP$_{\text{real}}$ $= \exists\mathbb{R} \subseteq \exists\mathbb{R}^{\Sigma} \subseteq$ PSPACE. We conjecture that all inclusions are strict. In this section, we will provide some arguments in favor of this.

We first observe that using summations we can quite easily solve PP-problems. In particular, we have:

▶ **Lemma 21.** $\mathsf{NP}^{\mathsf{PP}} \subseteq \exists\mathbb{R}^{\Sigma}$.

Proof. The canonical $\mathsf{NP}^{\mathsf{PP}}$-complete problem E-MajSat is deciding the satisfiability of a formula

$$\psi : \exists x_1, \ldots, x_n : \#\{(y_1, \ldots, y_n) \in \{0,1\}^n \mid \phi(x,y) = 1\} \geq 2^{n-1},$$

i.e., deciding whether there is an assignment to the x-variables such that the resulting formula is satisfied by at least half of the assignments to the y-variables [18].

Let $X_1 \ldots X_n, Y_1 \ldots Y_n$ be real variables and $\phi^{\mathbb{R}}$ the arithmetization of ϕ. We build an equivalent $\exists\mathbb{R}^{\Sigma}$ instance as follows:

1. $X_i = 0 \vee X_i = 1$, $1 \leq i \leq n$, and
2. $\displaystyle\sum_{Y_1=0}^{1} \cdots \sum_{Y_n=0}^{1} \phi^{\mathbb{R}}(X,Y) \geq 2^{n-1}$.

Then this instance is satisfiable iff ψ is satisfiable because X_i are existentially chosen and constraint to be Boolean and $\sum_{Y_1=0}^{1} \cdots \sum_{Y_n=0}^{1} \phi^{\mathbb{R}}(X,Y)$ is exactly the number of satisfying assignments to the Y-variables. ◀

Similarly to $\exists\mathbb{R} = \mathsf{NP}_{\mathrm{real}}$, we can also characterize $\exists\mathbb{R}^{\Sigma}$ using a machine model instead of a closure of a complete problem under polynomial time many one reductions. For this we define a $\mathsf{NP}_{\mathrm{real}}^{\mathsf{VNP}_{\mathbb{R}}}$ machine to be an $\mathsf{NP}_{\mathrm{real}}$ machine with a $\mathsf{VNP}_{\mathbb{R}}$ oracle, where $\mathsf{VNP}_{\mathbb{R}}$ denotes Valiant's NP over the reals. Since $\mathsf{VNP}_{\mathbb{R}}$ is a family of polynomials, the oracle allows us to evaluate a family of polynomials, for example the permanent, at any real input[8]. The two lemmas below demonstrate that $\mathsf{NP}_{\mathrm{real}}^{\mathsf{VNP}_{\mathbb{R}}}$ coincides with $\exists\mathbb{R}^{\Sigma}$ which strengthens Lemma 21 that $\mathsf{NP}^{\mathsf{PP}} \subseteq \exists\mathbb{R}^{\Sigma}$ and characterizes Σ-ETR in terms of complexity classes over the reals.

▶ **Lemma 22.** *Σ-ETR $\in \mathsf{NP}_{real}^{\mathsf{VNP}_{\mathbb{R}}}$. This also holds if the NP_{real} machine is only allowed to call its oracle once.*

▶ **Lemma 23.** *Σ-ETR is hard for $\mathsf{NP}_{real}^{\mathsf{VNP}_{\mathbb{R}}}$.*

▶ **Theorem 24.** *Σ-ETR is complete for $\mathsf{NP}_{real}^{\mathsf{VNP}_{\mathbb{R}}}$. Thus, $\exists\mathbb{R}^{\Sigma} = \mathsf{NP}_{real}^{\mathsf{VNP}_{\mathbb{R}}}$.*

Proof idea. To prove Lemma 22, we show a normal form for Σ-ETR instances such that all polynomials contained in it are of the form $\sum_{Y \in \{0,1\}^m} p(X,Y)$ where p does not contain any unary sums. Then we show how to translate formulas in this normal form into a real word-RAM with oracle access. For the hardness results of Lemma 23, we encode the real word-RAM computations into an ETR-instance, where the oracle calls (which w.l.o.g. can be assumed to be calls to the permanent) are simulated by the summation operator. ◀

8.2 Reasoning about Probabilities in Small Models

In this section, we employ the satisfiability problems for languages of the causal hierarchy. The problem $\textsc{Sat}_{sm,prob}^{poly\langle\Sigma\rangle}$ is defined like $\textsc{Sat}_{prob}^{poly\langle\Sigma\rangle}$, but in addition we require that a satisfying distribution has only polynomially large support, that is, only polynomially many entries in the exponentially large table of probabilities are nonzero. Formally we can achieve this by extending an instance with an additional unary input $p \in \mathbb{N}$ and requiring that the satisfying distribution has a support of size at most p. The membership proofs of $\textsc{Sat}_{prob}^{poly}$ in NP and

[8] See the full version for an overview of the relevant definitions.

in $\exists\mathbb{R}$, respectively, by [11], [17], and [21] rely on the fact that the considered formulas have the small model property: If the instance is satisfiable, then it is satisfiable by a small model. For $\mathrm{SAT}_{prob}^{poly\langle\Sigma\rangle}$, this does not seem to be true because we can directly force any model to be arbitrarily large, e.g., by encoding the additional parameter p above in binary or by enforcing a uniform distribution using $\sum_{x_1} \dots \sum_{x_n} (P(X_1{=}x_1, \dots, X_n{=}x_n) - P(X_1{=}0, \dots, X_n{=}0))^2 = 0$. Thus, we have to explicitly require that the models are small, yielding the problem $\mathrm{SAT}_{sm,prob}^{poly\langle\Sigma\rangle}$. Formally, we use the following:

▶ **Definition 25.** *The decision problems* $\mathrm{SAT}_{sm,prob}^{poly\langle\Sigma\rangle}$ *take as input a formula* $\varphi \in \mathcal{L}^{poly\langle\Sigma\rangle}$ *and a unary encoded number* $p \in \mathbb{N}$ *and ask whether there exists a model* $\mathfrak{M} = (\{X_1,\dots,X_n\}, P)$ *such that* $\mathfrak{M} \models \varphi$ *and* $\#\{(x_1,\dots,x_n) : P(X_1{=}x_1,\dots,X_n{=}x_n) > 0\} \leq p$.

It turns out that $\mathrm{SAT}_{sm,prob}^{poly\langle\Sigma\rangle}$ is a natural complete problem for $\exists\mathbb{R}^{\Sigma}$:

▶ **Theorem 26.** *The decision problem* $\mathrm{SAT}_{sm,prob}^{poly\langle\Sigma\rangle}$ *is complete for* $\exists\mathbb{R}^{\Sigma}$.

Proof idea. To show the containment of $\mathrm{SAT}_{sm,prob}^{poly\langle\Sigma\rangle}$ in $\exists\mathbb{R}^{\Sigma}$, we first show a normal form that every probability occurring in the input instance contains all variables. Then we have to use the exponential sum and the polynomially many variables to "built" a probability distribution with polynomial support. The lower bound follows from reducing from a restricted Σ-ETR-instance. ◀

9 Discussion

Traditionally, ETR has been used to characterize the complexity of problems from geometry and real optimization. It has recently been used to characterize probabilistic satisfiability problems, which play an important role in AI, see e.g. [21, 31]. We have further investigated the recently defined class succ-$\exists\mathbb{R}$, characterized it in terms of real word-RAMs, and shown the existence of further natural complete problems. Moreover, we defined a new class $\exists\mathbb{R}^{\Sigma}$ and also gave natural complete problems for it.

The studied summation operators allow the encoding of exponentiation, but only with integer bases, so they do not affect the decidability, unlike Tarski's exponential function [29].

Schäfer and Stefankovic [27] consider extensions of ETR where we have a constant number of alternating quantifiers instead of just one existential quantifier. By the work of Grigoriev and Vorobjov [14], these classes are all contained in PSPACE. Can we prove a real version of Toda's theorem [30]? Are these classes contained in $\exists\mathbb{R}^{\Sigma}$?

References

1 Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. In *Proc. of the 50th ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 65–73, 2018. doi:10.1145/3188745.3188868.

2 Mikkel Abrahamsen, Linda Kleist, and Tillmann Miltzow. Training neural networks is $\exists\mathbb{R}$-complete. In *Proc. Advances in Neural Information Processing Systems (NeurIPS 2021)*, pages 18293–18306, 2021.

3 Elias Bareinboim, Juan D. Correa, Duligur Ibeling, and Thomas Icard. *On Pearl's Hierarchy and the Foundations of Causal Inference*, pages 507–556. Association for Computing Machinery, New York, NY, USA, 2022. doi:10.1145/3501714.3501743.

4 Vittorio Bilò and Marios Mavronicolas. $\exists\mathbb{R}$-complete decision problems about symmetric Nash equilibria in symmetric multi-player games. In *Proc. 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

5 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

6 Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Science & Business Media, 2000.

7 John Canny. Some algebraic and geometric computations in PSPACE. In *Proc. of the 20th ACM Symposium on Theory of Computing (STOC 1988)*, pages 460–467. ACM, 1988. doi:10.1145/62212.62257.

8 Jean Cardinal. Computational geometry column 62. *ACM SIGACT News*, 46(4):69–78, 2015. doi:10.1145/2852040.2852053.

9 Arnaud Durand, Miika Hannula, Juha Kontinen, Arne Meier, and Jonni Virtema. Probabilistic team semantics. In *Foundations of Information and Knowledge Systems: 10th International Symposium, FoIKS 2018, Proceedings 10*, pages 186–206. Springer, 2018. doi:10.1007/978-3-319-90050-6_11.

10 Jeff Erickson, Ivor Van Der Hoog, and Tillmann Miltzow. Smoothing the gap between NP and ER. *SIAM Journal on Computing*, pages FOCS20–102, 2022.

11 Ronald Fagin, Joseph Y Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1-2):78–128, 1990. doi:10.1016/0890-5401(90)90060-U.

12 Jugal Garg, Ruta Mehta, Vijay V Vazirani, and Sadra Yazdanbod. ∃ℝ-completeness for decision versions of multi-player (symmetric) Nash equilibria. *ACM Transactions on Economics and Computation (TEAC)*, 6(1):1–23, 2018. doi:10.1145/3175494.

13 George Georgakopoulos, Dimitris Kavvadias, and Christos H. Papadimitriou. Probabilistic satisfiability. *Journal of Complexity*, 4(1):1–11, 1988. doi:10.1016/0885-064X(88)90006-4.

14 Dima Grigoriev and Nicolai N. Vorobjov Jr. Solving systems of polynomial inequalities in subexponential time. *J. Symb. Comput.*, 5(1/2):37–64, 1988. doi:10.1016/S0747-7171(88)80005-1.

15 Miika Hannula, Minna Hirvonen, Juha Kontinen, Yasir Mahmood, Arne Meier, and Jonni Virtema. Logics with probabilistic team semantics and the boolean negation. *arXiv preprint arXiv:2306.00420*, 2023. doi:10.48550/arXiv.2306.00420.

16 Miika Hannula, Juha Kontinen, Jan Van den Bussche, and Jonni Virtema. Descriptive complexity of real computation and probabilistic independence logic. In *Proc. of the 35th ACM/IEEE Symposium on Logic in Computer Science (LICS 2020)*, pages 550–563, 2020. doi:10.1145/3373718.3394773.

17 Duligur Ibeling and Thomas Icard. Probabilistic reasoning across the causal hierarchy. In *Proc. 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 10170–10177. AAAI Press, 2020. doi:10.1609/AAAI.V34I06.6577.

18 Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998. doi:10.1613/JAIR.505.

19 Meena Mahajan. Algebraic complexity classes. *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 51–75, 2014.

20 Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B*, 103(1):114–143, 2013. doi:10.1016/J.JCTB.2012.09.004.

21 Milan Mossé, Duligur Ibeling, and Thomas Icard. Is causal reasoning harder than probabilistic reasoning? *The Review of Symbolic Logic*, pages 1–26, 2022.

22 Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986. doi:10.1016/0004-3702(86)90031-7.

23 Judea Pearl. *Causality.* Cambridge University Press, 2009.

24 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. Part I: Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals. *Journal of symbolic computation*, 13(3):255–299, 1992. doi:10.1016/S0747-7171(10)80003-3.

25 Marcus Schaefer. Complexity of some geometric and topological problems. In *Proc. International Symposium on Graph Drawing (GD 2009)*, pages 334–344. Springer, 2009. doi:10.1007/978-3-642-11805-0_32.

26 Marcus Schaefer, Jean Cardinal, and Tillmann Miltzow. The existential theory of the reals as a complexity class: A compendium. *arXiv preprint*, 2024. doi:10.48550/arXiv.2407.18006.

27 Marcus Schaefer and Daniel Štefankovič. Beyond the existential theory of the reals. *Theory of Computing Systems*, 68(2):195–226, 2024. doi:10.1007/S00224-023-10151-X.

28 Ilya Shpitser and Judea Pearl. Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, 9:1941–1979, 2008. doi:10.5555/1390681.1442797.

29 Alfred Tarski. A decision method for elementary algebra and geometry. *Journal of Symbolic Logic*, 14(3), 1949.

30 Seinosuke Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.

31 Benito van der Zander, Markus Bläser, and Maciej Liśkiewicz. The hardness of reasoning about probabilities and causality. In *Proc. Joint Conference on Artificial Intelligence (IJCAI 2023)*, 2023.

Routing from Pentagon to Octagon Delaunay Graphs

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Jean-Lou De Carufel
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

John Stuart
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Abstract

The standard Delaunay triangulation is a geometric graph whose vertices are points in the plane, and two vertices share an edge if they lie on the boundary of an empty disk. If the disk is replaced with a homothet of a fixed convex shape C, then the resulting graph is called a C-Delaunay graph. We study the problem of local routing in C-Delaunay graphs where C is a regular polygon having five to eight sides. In particular, we generalize the routing algorithm of Chew for square-Delaunay graphs (Chew. *SCG 1986*, 169–177) in order to obtain the following approximate upper bounds of 4.640, 6.429, 8.531 and 4.054 on the spanning and routing ratios for pentagon-, hexagon-, septagon-, and octagon-Delaunay graphs, respectively. The exact expression for the upper bounds of the routing ratio is

$$\Psi(n) := \begin{cases} \sqrt{1 + ((\cos(2\pi/n) + n - 1)/\sin(2\pi/n))^2} & \text{if } n \in \{5, 6, 7\}, \\ \sqrt{1 + ((\cos(\pi/8)\cos(3\pi/8) + 3)/(\cos(\pi/8)\sin(3\pi/8)))^2} & \text{if } n = 8. \end{cases}$$

We show that these bounds are tight for the output of our routing algorithm by providing a point set where these bounds are achieved. We also include lower bounds of 1.708 and 1.995 on the spanning and routing ratios of the pentagon-Delaunay graph.

Our upper bounds yield a significant improvement over the previous routing ratio upper bounds for this problem, which previously sat at around 400 for the pentagon, septagon, and octagon as well as 18 for the hexagon. Our routing ratios also provide significant improvements over the previously best known spanning ratios for pentagon-, septagon- and octagon-Delaunay graphs, which were around 45.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Geometric Spanners, Generalized Delaunay Graphs, Local Routing Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.14

1 Introduction

A geometric graph is a weighted graph whose vertices are points in the plane and edges are line segments weighted with the Euclidean distance between their endpoints. Two of the main distance-preserving properties of a graph are the spanning ratio and routing ratio. The spanning ratio of a pair of points is the ratio of the shortest path between them in the graph divided by their Euclidean distance, and the spanning ratio of a graph is the maximum spanning ratio over all pairs of points [11]. On the other hand, the routing ratio is defined similarly, except that the path is usually computed locally with only information of the current vertex's neighbourhood. Since a routing ratio is based on an algorithm that finds a path and the spanning ratio is based on the existence of a path, the spanning ratio of any graph is a lower bound on the routing ratio.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 14; pp. 14:1–14:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider variants of standard Delaunay triangulations, which are geometric graphs with an edge between two points if there exists a disk with the endpoints on its boundary and no vertices in its interior. The spanning ratio of the Delaunay triangulation is known to be between 1.5932 [14] and 1.998 [13], however the exact value still remains unknown. The gap is even larger for the routing ratio, lying somewhere between 1.70 [2] and 3.56 [1]. Many papers study the related graphs that result from replacing the disk with a homothet of a fixed convex shape C, resulting in C-Delaunay graphs. Chew [8] proved that square-Delaunay graphs have a spanning ratio of at most $\sqrt{10}$ by giving a local routing algorithm. Subsequently, Chew [9] adapted his algorithm to equilateral triangle-Delaunay graphs to find a spanning ratio of 2, however the adapted algorithm was no longer a routing algorithm. In fact, Bose et al. [6] showed that the routing ratio of the equilateral triangle-Delaunay graph is exactly $\frac{5}{\sqrt{3}}$, showing the first separation between the spanning ratio and routing ratio. By generalizing Chew's algorithm, Bose et al. [2] were then able to show that the standard Delaunay triangulation has a routing ratio of at most 5.90 which was an improvement on the previously known upper bound of 15.48 [5]. Currently, the best-known bound is 3.56[1]. In this paper, we show that Chew's algorithm can be further generalized to pentagon-, hexagon-, septagon-, and octagon-Delaunay graphs to obtain routing ratios of 4.640, 6.429, 8.531 and 4.054, respectively.

The hexagon-Delaunay graph is known to have a tight spanning ratio of 2 [12], however less is known about Delaunay graphs based on pentagons, septagons and octagons. With the exception of the hexagon-Delaunay graph, our routing ratio upper bounds yield a significant improvement over the previous best spanning ratio upper bounds. Bose et al. [4] give a spanning ratio upper bound for any C-Delaunay graph, where C is any convex shape. In particular, their bound is based intuitively on the thinness of C, which is essentially measured by the ratio of the perimeter to the width of C. For example, this ratio is π when C is a disk. Furthermore, by the construction of the paths from Bose et al. [4] and Perkovic et al. [12], it is possible to route using the algorithm of Bose and Morin [7] with a constant routing ratio of 9 times the spanning ratio. For each polygon, we compare our contribution to the previous best known upper bound in Table 1. We also prove lower bounds of 1.708 and 1.995 on the spanning and routing ratios of the pentagon-Delaunay graph in the appendix.

Table 1 Comparison to previously best-known upper bounds on the spanning and routing ratio of the C-Delaunay graph.

C	Spanning Ratio	Routing Ratio	Our Routing and Spanning Ratio
Triangle	2[9]	$5/\sqrt{3}$[6]	
Square	$\sqrt{4+2\sqrt{2}}$ [3]	$\sqrt{10}$[8]	
Pentagon	≈ 45[4]	≈ 405[7]	≈ 4.640
Hexagon	2[12]	18[7]	≈ 6.429
Septagon	≈ 45[4]	≈ 405[7]	≈ 8.531
Octagon	≈ 43[4]	≈ 387[7]	≈ 4.054
Circle	≈ 1.998[13]	≈ 3.56[1]	

2 Preliminaries

We denote the line segment between points u, v as uv, and the Euclidean length of uv is denoted $|uv|$. For a path $\mathcal{P}$ in the plane, denote $|\mathcal{P}|$ as the length of the path. If paths $\mathcal{P}, \mathcal{Q}$ share an endpoint, then $\mathcal{P} + \mathcal{Q}$ denotes their concatenation. Next, for $a, b, c \in \mathbb{R}^2$, we define

$\angle abc$ as the angle from ab to bc clockwise around b. The x and y coordinates of $a \in \mathbb{R}^2$ are denoted $x(a), y(a)$ respectively. For two vertices u, v in a geometric graph G, the length of the shortest path from u to v in G is denoted $d_G(u, v)$. Then for a constant $c \geq 1$, G is said to be a c-spanner if for all points u, v in G, we have $d_G(u, v) \leq c|uv|$. The spanning ratio of G is the least c for which G is a c-spanner. The spanning ratio of a class of graphs $\mathcal{G}$ is the least c for which all graphs in $\mathcal{G}$ are c-spanners. A *constant* spanner is a c-spanner where c is a constant.

We make the assumption that the graph is embedded on a polynomial-sized grid and therefore specifying the coordinates of a vertex in $V(G)$ requires $O(\log(|V(G)|))$ bits. Formally, a m-memory local routing algorithm is a function that takes as input $(s, N(s), t, M)$, and outputs some memory M' and a vertex $p \in N(s)$ where s is the current vertex, $N(s)$ is the neighbourhood of s, t is the destination, and both M, M' are bit-strings of length m. An algorithm is said to be c-competitive for a family of geometric graphs $\mathcal{G}$ if the path output by the algorithm for any pair of vertices $s, t \in V(G)$ for $G \in \mathcal{G}$ has length at most $c|st|$. The routing ratio of an algorithm is the least c for which the algorithm is c-competitive for $\mathcal{G}$. Note that the routing ratio is an upper bound on the spanning ratio.

For $n \in \{5, 6, 7, 8\}$, let $\bigcirc_n$ denote a regular n-gon in the plane. Every time we mention an n-gon, it is assumed to be a scaled translate of $\bigcirc_n$. Note that rotations are not permitted. We refer to the boundary of any n-gon C as ∂C, and to the interior as $\text{int}(C)$. We make the *general position* assumptions that no two points are on a line parallel to a side of $\bigcirc_n$, that neither coordinate axis is parallel to a side of $\bigcirc_n$, and that no four points lie on ∂C for some n-gon C. For two points $a, b \in \partial C$, define $\text{Arc}(C, a, b)$ to be the clockwise portion of ∂C from a to b. Let S be a set of points in the plane.

▶ **Definition 1.** *For $a, b \in S$, an edge ab satisfies the empty-$\bigcirc_n$ property with respect to S if there exists an n-gon C with $a, b \in \partial C$ and $S \cap \text{int}(C) = \emptyset$.*

▶ **Definition 2.** *A $\bigcirc_n$-Delaunay graph of S is a maximal planar graph on S such that every edge satisfies the empty-$\bigcirc_n$ property with respect to S. By maximal, we mean that no more edges satisfying the empty-$\bigcirc_n$ property can be added.*

Note that specifying maximality in Definition 2 guarantees that every bounded face is a triangle [4]. Let u, v be two points in the plane that satisfy the general position assumption. Then $\text{Boundary}(u, v)$ denotes the set of n-gons C such that $u, v \in \partial C$. Also for any homothet C, define the point $\text{Center}(C)$ to be the point in C equidistant from all vertices of C. Furthermore, denote $\text{North}(C)$ to be the vertex of C with the largest y-coordinate. Similarly, define $\text{East}(C)$, $\text{South}(C)$ and $\text{West}(C)$. For a set H of homothets of $\bigcirc_n$, let $\text{Center}(H) := \{\text{Center}(C) \mid C \in H\}$. Similarly, we define $\text{West}(H)$. For any homothet C of the n-gon $\bigcirc_n$, we label the vertices clockwise from $\text{West}(C)$ as $C^1, ..., C^n$.

3 Routing Ratio Upper Bound

Recall that

$$\Psi(n) := \begin{cases} \sqrt{1 + ((\cos(2\pi/n) + n - 1)/\sin(2\pi/n))^2} & \text{if } n \in \{5, 6, 7\}, \\ \sqrt{1 + ((\cos(\pi/8)\cos(3\pi/8) + 3)/(\cos(\pi/8)\sin(3\pi/8)))^2} & \text{if } n = 8. \end{cases}$$

The goal of this section is to prove the following theorem.

▶ **Theorem 3.** *The routing ratio of the $\bigcirc_5$-Delaunay graph is at most $\Psi(5) \approx 4.64$.*

3.1 Local Routing Algorithm

We present Algorithm 1, which is a $O(\log(|V(G)|))$-memory local routing algorithm for $\bigcirc_n$-Delaunay graphs generalizing Chew's local routing algorithm [8]. Without loss of generality, we assume that the start vertex s and destination t satisfy $y(s) = y(t)$ and $x(s) < x(t)$. The location of s is stored in memory for each step. In addition, we will assume that all edges of the convex hull are present in the $\bigcirc_n$-Delaunay graph of S. Then in Section 3.2, we describe how Algorithm 1 can be modified to handle routing when the convex hull is not present.

The intuition behind Algorithm 1 is that when the current vertex is p_i, the next vertex p_{i+1} is restricted to one of the vertices of the rightmost triangle T_i of the graph containing vertex p_i and intersecting st. Note that the two neighbours of p_i under consideration are on opposite sides of st. Then, consider the empty n-gon C_i corresponding to T_i. We partition the boundary of C_i into two arcs by splitting at its west point w_i and its rightmost intersection with st, denoted t_i. If p_i is in the upper arc, we choose the clockwise neighbour, otherwise we choose the counterclockwise neighbour. A trace of Algorithm 1 is illustrated in Figure 1. Note that the triangles $T_0, ..., T_k$ are ordered from left to right along st, so the algorithm terminates. We assume that there are $k + 1$ edges in the path output by Algorithm 1. Note that the last edge $p_k t$ in Algorithm 1 may appear to be a separate case from case b in Algorithm 1, but we can avoid analyzing it separately by viewing t as both above and below st. The important detail in the analysis is that t is in the same portion of ∂C_k as p_k (either $p_k, t \in \mathrm{Arc}(C_k, w_k, t_k)$ or $p_k, t \in \mathrm{Arc}(C_k, t_k, w_k)$).

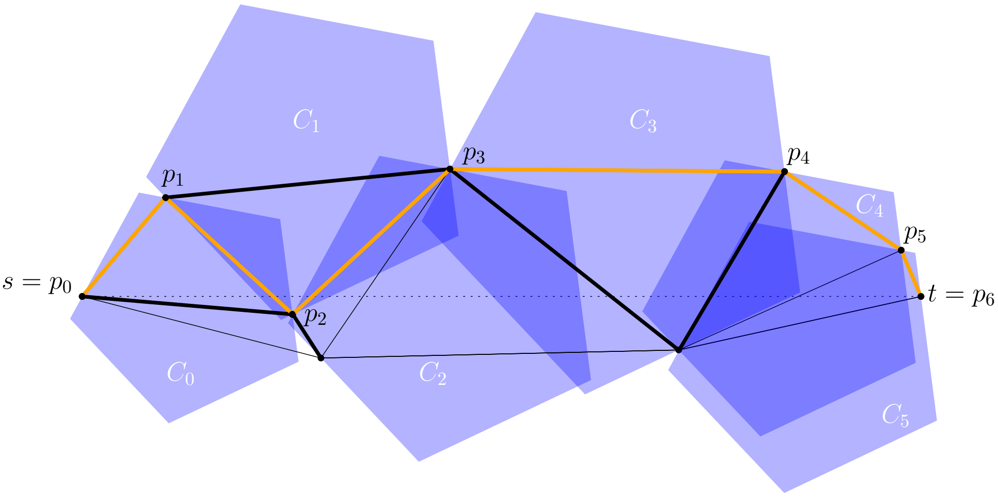


Figure 1 Trace of Algorithm 1. In this case, $k = 5$ and $n = 5$. The orange line is the path chosen by Algorithm 1 and the thick black edges represent the other edge ($p_i a$ or $p_i b$) considered in step b.

When u, v are in general position, the set Center(Boundary(u, v)) is analogous to the perpendicular bisector of uv when the n-gon $\bigcirc_n$ is replaced with a disk. For this reason, we will refer to Center(Boundary(u, v)) as Bisector(u, v). In Lemma 2.2.1.1 of [10], Ma shows that for any regular n-gon, Bisector(u, v) is a polygonal chain completed with two rays at the ends. In this way, Bisector(u, v) partitions $\mathbb{R}^2$ into two half-spaces (see Figure 2).

When $y(v) > y(u)$, then there is a natural ordering of the points in Bisector(u, v) from left to right. By convention, for any $u, v \in \mathbb{R}^2$ in general position, we say that the point Bisector$(u, v) \cap \mathrm{Arc}(C, u, v)$ is to the left of the point Bisector$(u, v) \cap \mathrm{Arc}(C, v, u)$. This is extended to an ordering on all the points of Bisector(u, v). Note that with this convention, Bisector(u, v) does not have the same ordering as Bisector(v, u). For example, this convention

Algorithm 1 Local Routing algorithm in $\bigcirc_n$-Delaunay triangulation.

```
Data: Two points $s, t \in S$ (w.l.o.g. $y(s) = y(t)$ and $x(s) < x(t)$)
Result: Vertices $s = p_0, ..., t = p_{k+1}$ forming a path in $\bigcirc_n$-Delaunay graph of $S$
Set $i \leftarrow 0$ and $p_i \leftarrow s$;
while $p_i \neq t$ do
    (a) If $t$ is a neighbour of $p_i$, set $p_{i+1} \leftarrow t$.
    (b) Otherwise, let $T_i$ be the rightmost triangle in the graph intersecting $st$ with
        vertex $p_i$. Let $a, b$ be the other vertices of $T_i$ above and below $st$, respectively.
        Let $C_i$ be the empty $n$-gon with $p_i, a, b \in \partial C_i$. Let $w_i :=$ West$(C_i)$ and $t_i$ be
        the intersection of $C_i$ with $st$ closest to $t$.
        (1) If $p_i \in$ Arc$(C_i, w_i, t_i)$, then set $p_{i+1} \leftarrow a$ and $i \leftarrow i + 1$.
        (2) Else, set $p_{i+1} \leftarrow b$ and $i \leftarrow i + 1$.
end
```

tells us that in Figure 2, Center(C_{i-1}) is to the left of Center(C_i) on Bisector(q, p_i), whereas Center(C_{i-1}) is to the right of Center(C_i) on Bisector(p_i, q). Then the following remark is based on Ma's plane sweep algorithm [10] which produces the vertices of Bisector(u, v).

▶ Remark 4 ([10]). Let $u, v \in \mathbb{R}^2$ be in general position with $y(v) > y(u)$. Then for any $C, C' \in$ Boundary(u, v) we have that $\angle$South(C)Center$(C)v \geq \angle$South(C')Center$(C')v$ provided that Center(C) is to the left of Center(C') on Bisector(u, v).

In the following lemma, we describe the structure of the path output by Algorithm 1.

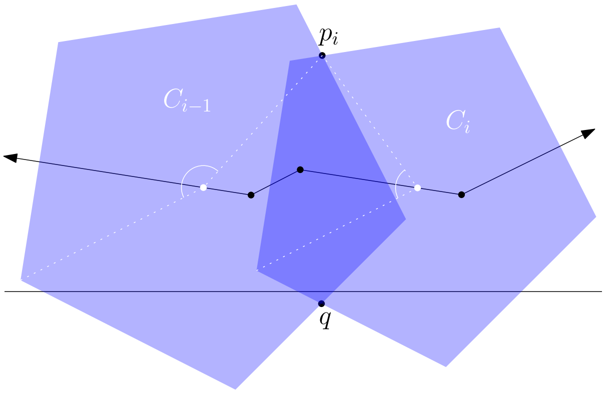


Figure 2 The black polygonal chain is Bisector(q, p_i). The white points are the centers of the 5-gons C_{i-1} and C_i.

▶ **Lemma 5.** *Let $i \in \{1, ..., k\}$. If edges $p_{i-1}p_i$ and p_ip_{i+1} both use case b1 in Algorithm 1, then $\angle w_{i-1}$Center$(C_{i-1})p_i \geq \angle w_i$Center$(C_i)p_i$. If instead both edges use case b2 in Algorithm 1, then $\angle p_i$Center$(C_{i-1})w_{i-1} \geq \angle p_i$Center$(C_i)w_i$.*

Proof. Assume without loss of generality that edges $p_{i-1}p_i$ and p_ip_{i+1} were chosen using case b1 in Algorithm 1. Then p_i is above st. We will first consider the case where T_{i-1} and T_i share an edge, denoted p_iq. Since p_i is above st, then q is below st. Refer to Figure 2. Both Center(C_{i-1}) and Center(C_i) lie on Bisector(q, p_i), however it remains to establish their relative position. Then by Remark 4, the result follows if we show that Center(C_{i-1}) is to the left of Center(C_i). Define $L := \{$Center$(C) \mid C \in$ Boundary(p_i, q) and $p_{i-1} \in$ int$(C)\}$ and

$R := \{\text{Center}(C) \mid C \in \text{Boundary}(p_i, q) \text{ and } p_{i+1} \in \text{int}(C)\}$. Since $p_{i-1} \in \text{Arc}(C_{i-1}, q, p_i)$, then L propagates from Center(C_{i-1}) to the left. Similarly, R propagates from Center(C_i) to the right since $p_{i+1} \in \text{Arc}(C_i, p_i, q)$.If Center$(C_{i-1})$ is to the right of Center(C_i), then $L \cup R = \text{Bisector}(q, p_i)$. However, Center$(C_{i-1})$ and Center(C_i) are both examples of points in Bisector(q, p_i) but not in $L \cup R$. Therefore Center(C_{i-1}) is to the left of Center(C_i), hence $\angle\text{South}(C_{i-1})\text{Center}(C_{i-1})p_i \geq \angle\text{South}(C_i)\text{Center}(C_i)p_i$. Then we obtain the desired inequality by remarking that p_i is not on Arc$(C_r, \text{South}(C_r), w_r)$ for $r \in \{i-1, i\}$ and also the angle $\angle\text{South}(C)\text{Center}(C)\text{West}(C)$ is constant for all homothets C of $\bigcirc_n$.

If T_i and T_{i-1} do not share an edge, then we use this argument on all the triangles between T_{i-1} and T_i. The result follows since inequality is transitive. ◀

Next, we define the worst-case n-gons, shown in Figure 3.

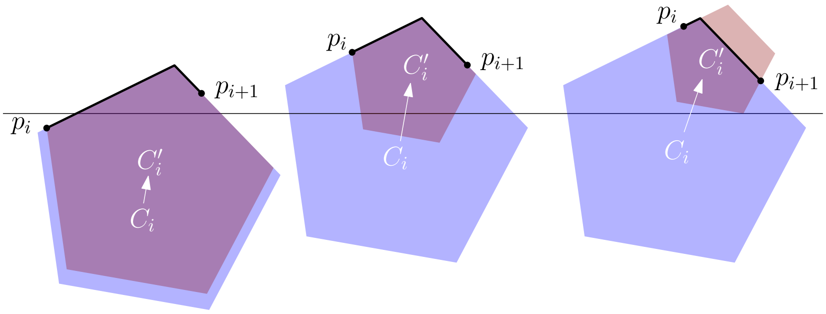


Figure 3 The original 5-gons, C_i, are blue, and the worst-case 5-gons, C_i' from Definition 6, are purple. In all examples, p_{i+1} is above st. Left: West$(C_i') = p_i$. Middle: West$(C_i') = p_i$. Right: South$(C_i'), s, t$ are collinear.

▶ **Definition 6** (Worst-Case n-gons). *Let $i \in \{0, ..., k\}$ and suppose p_{i+1} is above st. Start with an n-gon $C = C_i$, then move Center(C) left along Bisector(p_i, p_{i+1}) while keeping $C \in$ Boundary(p_i, p_{i+1}) until the points South$(C), s, t$ are collinear, or $p_i =$ West(C). The resulting n-gon is denoted by C_i'. If instead p_{i+1} is below st, then move Center(C) right along Bisector(p_i, p_{i+1}) while keeping $C \in$ Boundary(p_i, p_{i+1}) until the points North$(C), s, t$ are collinear, or $p_i =$ West(C). The resulting n-gon is again denoted by C_i'.*

To shorten notation, denote $w_i' := \text{West}(C_i')$ for $i \in \{0, ..., k\}$. A similar statement to that of Lemma 5 can be made about the worst-case n-gons:

▶ **Lemma 7.** *Let $i \in \{1, ..., k\}$. If edges $p_{i-1}p_i$ and p_ip_{i+1} both use case b1 in Algorithm 1, then $\angle w_{i-1}'\text{Center}(C_{i-1}')p_i \geq \angle w_i'\text{Center}(C_i')p_i$. If instead both edges use case b2 in Algorithm 1, then $\angle p_i\text{Center}(C_{i-1}')w_{i-1}' \geq \angle p_i\text{Center}(C_i')w_i'$.*

Proof. Let $i \in \{1, ..., k\}$ and assume edges $p_{i-1}p_i$ and p_ip_{i+1} both use case b1 in Algorithm 1. By Lemma 5, we have $\angle w_{i-1}\text{Center}(C_{i-1})p_i \geq \angle w_i\text{Center}(C_i)p_i$. Since p_i is above st, then the construction of Definition 6 guarantees that $\angle w_{i-1}\text{Center}(C_{i-1})p_i \leq \angle w_{i-1}'\text{Center}(C_{i-1}')p_i$. Similarly, p_{i+1} is above st, meaning that $\angle w_i'\text{Center}(C_i')p_i \leq \angle w_i\text{Center}(C_i)p_i$. Combining inequalities yields the result. Proving the case when both edges use case b2 in Algorithm 1 is similar. ◀

Using the same point set as in Figure 1, the trace of Algorithm 1 is shown in Figure 4 with worst-case n-gons.

Now we define the wedge W_p to be the area swept by stretching $\bigcirc_n$ with its west point on p. More precisely, $W_p := \{v \in \mathbb{R}^2 \mid \exists \text{ homothet } C \text{ of } \bigcirc_n \text{ such that } p = \text{West}(C) \text{ and } v \in C\}$.

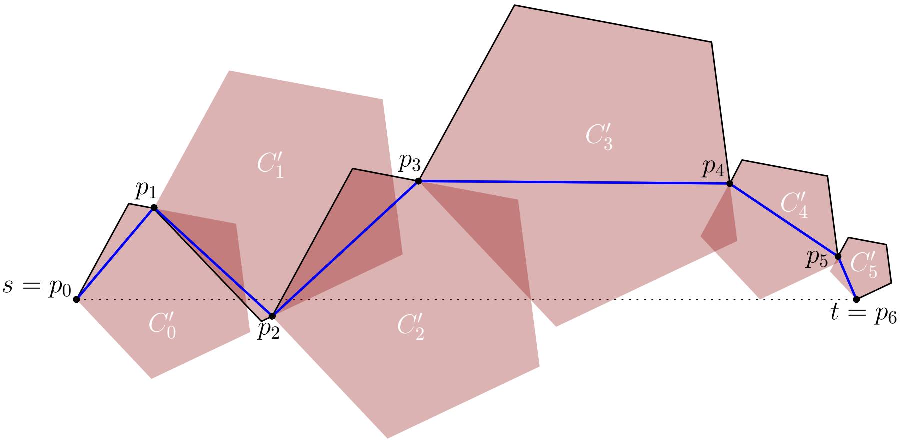


Figure 4 Trace of Algorithm 1 with worst-case 5-gons.

▶ **Lemma 8.** *For $i \in \{1, ..., k\}$, we have $w'_i \in W_{w'_{i-1}}$.*

Proof. Suppose p_i is above st. If $p_i = w'_i$, then w'_i is on the boundary of C'_{i-1}, hence $w'_i \in C_{w'_{i-1}}$. If on the other hand $p_i \neq w'_i$, by Definition 6 we must have $p_i \in \text{Arc}(C'_i, w'_i, \text{South}(C'_i))$ and $\text{South}(C'_i)$ is on st. Suppose for now that $\text{South}(C'_{i-1})$ is also on st. Define the set of homothets

$$L := \{C \mid \text{South}(C), s, t \text{ are collinear and } p_i \in \text{Arc}(C, \text{West}(C), \text{South}(C))\}.$$

Then we will prove the following claim, illustrated in Figure 5.

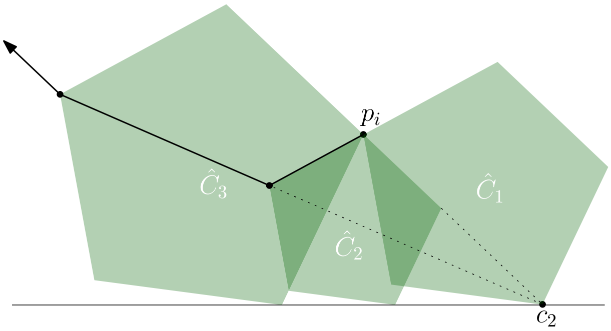


Figure 5 $\text{West}(L)$ is the black line, and c_2 is the homothety center relating $\hat{C}_2$ and $\hat{C}_3$. Segments $\text{West}(\hat{C}_2)\text{West}(\hat{C}_3)$ and $c_2\text{West}(\hat{C}_2)$ lie on the same line. In this example, $\sigma = 4$ and $m = 2$.

▷ Claim. $\text{West}(L)$ is a polygonal chain connected to a ray.

Firstly, suppose $1 < \sigma < n$ such that $C^\sigma = \text{South}(C)$. Then for $j \in \{1, ..., \sigma - 1\}$, define $\hat{C}_j$ to be the unique homothet of $\bigcirc_n$ where $p_i = \hat{C}^j_j$ and $\hat{C}^\sigma_j$ is collinear with st. By definition, for $j \in \{1, ..., \sigma - 1\}$, we have $\text{West}(\hat{C}_j) \in \text{West}(L)$.

Next, for $j \in \{1, ..., \sigma - 2\}$, the n-gons $\hat{C}_j$ and $\hat{C}_{j+1}$ are related by a homothety whose center c_j lies on the intersection of lines given by extending the segments st and $\hat{C}^j_j \hat{C}^{j+1}_j$. Furthermore, for any $C \in L$ with p_i on $C^j C^{j+1}$, the homothety relating $\hat{C}_j$ and C has the same center, c_j. Therefore the point $\text{West}(C)$ lies on the segment $\text{West}(\hat{C}_j)\text{West}(\hat{C}_{j+1})$. On the other hand, for $C \in L$ with p_i on $C^{\sigma-1}C^\sigma$, the n-gons C and $\hat{C}_{\sigma-1}$ are related by a homothety whose center is South$(\hat{C}_{\sigma-1})$. Therefore the point West(C) must lie on the ray r starting at West$(\hat{C}_{\sigma-1})$ extending directly opposite South$(\hat{C}_{\sigma-1})$. We have shown that West(L) is the polygonal chain (West$(\hat{C}_1), ...,$ West$(\hat{C}_{\sigma-1})$) along with the ray r, so the proof of the claim is completed.

To establish a direction on West(L), which is homeomorphic to a ray by the claim, we choose the convention that p_i is the rightmost point. Then since the vertices C^j are labelled in the clockwise orientation in the claim, we have that for $C, C' \in L$, if West(C) is to the left of West(C') on West(L) then $\angle\text{West}(C)\text{Center}(C)p_i \geq \angle\text{West}(C')\text{Center}(C')p_i$. Therefore by Lemma 5, w'_i must be to the right of w'_{i-1} on West(L).

Finally, we will show that the slope of West(L) remains between the slope of $\bigcirc_n^1\bigcirc_n^n$ and $\bigcirc_n^1\bigcirc_n^2$. Recall the notation $\bigcirc_n^j$ denotes the j-th vertex clockwise around $\bigcirc_n$ where $\bigcirc_n^1 :=$ West$(\bigcirc_n)$. Let $1 < m < n$ such that $\bigcirc_n^m =$ North$(\bigcirc_n)$. We analyze West(L) in three portions.

Segment West$(\hat{C}_j)$West$(\hat{C}_{j+1})$ for $j \in \{1, ..., m-1\}$. Let $j \in \{1, ..., m-1\}$. Then by the homothety relating $\hat{C}_j$ and $\hat{C}_{j+1}$, the segment West$(\hat{C}_j)$West$(\hat{C}_{j+1})$ has slope equal to the slope of segment c_jWest$(\hat{C}_j)$. Since c_j lies to the left of the line by extending West$(\hat{C}_j)\hat{C}_j^2$, then the slope of c_jWest$(\hat{C}_j)$ is positive and less than the slope of $\bigcirc_n^1\bigcirc_n^2$.

Segment West$(\hat{C}_j)$West$(\hat{C}_{j+1})$ for $j \in \{m, ..., \sigma-2\}$. Suppose $j \in \{m, ..., \sigma-2\}$. Then by the homothety relating $\hat{C}_j$ and $\hat{C}_{j+1}$, the segment West$(\hat{C}_j)$West$(\hat{C}_{j+1})$ has slope equal to the slope of segment West$(\hat{C}_j)c_j$. Since c_j lies to the right of South$(\hat{C}_j)$, then the slope of West$(\hat{C}_j)c_j$ is negative and greater than or equal to the slope of $\bigcirc_n^1\bigcirc_n^\sigma$, which is always at least the slope of $\bigcirc_n^1\bigcirc_n^n$.

Ray r. The slope of ray r is West$(\hat{C}_{\sigma-1})c_{\sigma-1}$. Since $c_{\sigma-1}$ is South$(\hat{C}_{\sigma-1})$, then the slope of the ray is equal to the slope of $\bigcirc_n^1\bigcirc_n^\sigma$, which is again in the desired range.

Therefore the slope of each segment of West(L) is within the range given by the cone $W_{w'_{i-1}}$. Since w'_i is to the right on West(L), then we must have $w'_i \in W_{w'_{i-1}}$.

If now South(C'_{i-1}) is instead below st, then we define the homothet C' such that $p_i \in \partial C'$, the points s, South(C'), t are collinear, and Center(C'_{i-1}), Center(C'), p_i are also collinear. Since C' is fully contained in C'_{i-1}, then West$(C') \in W_{w'_{i-1}}$. Also, $\angle\text{West}(C')\text{Center}(C')p_i = \angle\text{West}(C'_{i-1})\text{Center}(C'_{i-1})p_i$, therefore the argument from above can show that $w'_i \in W_{\text{West}(C')}$, hence $w'_i \in W_{w'_{i-1}}$. ◀

Next we define a projection that depends on whether the point is above or below st.

▶ **Definition 9** (West Side Projection). *For a point p above st, let $\overline{p}$ be the intersection of st with the line passing through p with the same slope as $\bigcirc_n^1\bigcirc_n^2$. Similarly, when p is below st, let $\overline{p}$ be the intersection of st with the line passing through p with the same slope as $\bigcirc_n^1\bigcirc_n^n$.*

Using the projections, we define the two paths that we will call *snail paths* and that are used to bound the length of the path output by Algorithm 1.

▶ **Definition 10** (Snail Paths). *Let $a, b \in \mathbb{R}^2$ satisfy $y(a) = y(b)$. When $x(a) < x(b)$, define the n-gon C such that $b =$ South(C) and a, C^1, C^2 are collinear. Then $\diagup\!\!\sqsupset_{a,b} := aC^1 +$ Arc(C, C^1, b). Similarly, let C' be the n-gon such that $b =$ North(C') and a, C'^1, C'^n are collinear, and then $\diagdown\!\!\sqsupset_{a,b} := aC'^1 +$ Arc(C', b, C'^1). When $x(a) \geq x(b)$, we define $\diagup\!\!\sqsupset_{a,b}$ and $\diagdown\!\!\sqsupset_{a,b}$ to be empty paths.*

The shape of the snail path is very important as it will directly lead to the routing ratio in the proof of Theorem 3. Notice how the path in Figure 9 is arbitrarily close to $\diagup\!\!\sqsupset_{s,t}$.

▶ Remark 11. After fixing the orientation of the $\bigcirc_n$, there exists a constant $c > 1$ such that for any $a, b \in \mathbb{R}^2$ with $y(a) = y(b)$ and $x(a) < x(b)$, we have $|\nearrow\!\!\supset_{a,b}| = c|\overline{ab}|$. When $x(a) \geq x(b)$, then $|\nearrow\!\!\supset_{a,b}| = 0$. The same is true for $|\searrow\!\!\supset_{a,b}|$.

When $n = 5$, we prove $\Psi(5)$ is an upper bound on the constant c from Remark 11.

▶ **Lemma 12.** *Let $n = 5$ and $a, b \in \mathbb{R}^2$ with $y(a) = y(b)$ and $x(a) < x(b)$. Then* $\max(|\nearrow\!\!\supset_{a,b}|, |\searrow\!\!\supset_{a,b}|) \leq \Psi(5)|ab|$.

Proof. Let C be the 5-gon corresponding to $\nearrow\!\!\supset_{a,b}$, and let $\theta := \angle \text{West}(C)ab$. For now, assume $C^5 = \text{South}(C)$, meaning $\pi/5 \leq \theta$, and also $\theta \leq \pi/2$. Then the side length $|\text{West}(C)\text{South}(C)| = \frac{|ab|\sin(\theta)}{\sin(2\pi/5)}$ by the law of sines. Similarly, $|a\text{West}(C)| = \frac{|ab|\sin(2\pi/5+\theta)}{\sin(2\pi/5)}$. Since $|\nearrow\!\!\supset_{a,b}| = |a\text{West}(C)| + 4|\text{West}(C)\text{South}(C)|$, then we have

$$|\nearrow\!\!\supset_{a,b}| = \frac{|ab|\sin(2\pi/5+\theta)}{\sin(2\pi/5)} + 4\frac{|ab|\sin(\theta)}{\sin(2\pi/5)} \leq \Psi(5)|ab|,$$

where the last inequality follows from the analysis in the appendix. See Lemma 16. It is straightforward to verify that the claim still holds when $C^5 \neq \text{South}(C)$. The analysis for $\searrow\!\!\supset_{a,b}$ is symmetric. ◀

One useful tool that we will often use is a convex path bound from [2].

▶ **Observation 13** (Convex Path Bound). *Suppose two convex paths $\mathcal{P}_1, \mathcal{P}_2$ have the same endpoints a and b, and $\mathcal{P}_1$ is contained in the region formed by the simple polygon $\mathcal{P}_2 + ab$. Then $|\mathcal{P}_1| \leq |\mathcal{P}_2|$.*

Next, for $i \in \{0, ..., k\}$, we define the path $P_i := \overline{w'_i}w'_i + \text{Arc}(C'_i, w'_i, p_i)$. Paths of this form will be used in the following lemma for pentagons ($n = 5$).

▶ **Lemma 14.** *Let $n = 5$. For $1 \leq i \leq k$, we have*

$$|P_{i-1}| + |p_{i-1}p_i| \leq \Psi(5)|\overline{w'_{i-1}w'_i}| + |P_i|. \tag{1}$$

Furthermore, we have $|P_k| + |p_k t| \leq \Psi(5)|\overline{w'_k}, t|$.

Proof. For $i \in \{1, ..., k\}$, we will show that (1) holds using a case analysis. Without loss of generality, we will assume that p_i is above st for all cases. This is equivalent to assuming that the routing decision is case b1 in Algorithm 1, meaning that $p_{i-1} \in \text{Arc}(C'_{i-1}, w'_{i-1}, p_i)$. The arguments for when p_i is below st are symmetric. We will use the following shorthand: $s'_i := \text{South}(C'_i)$.

Case 1: p_{i-1} is above st and both s'_{i-1}, s'_i lie on st. See Figure 6. We split the snail path of C'_{i-1} up into several parts:

$$|\nearrow\!\!\supset_{\overline{w'_{i-1}},s'_{i-1}}| = |P_{i-1}| + |\text{Arc}(C'_{i-1}, p_{i-1}, p_i)| + |\text{Arc}(C'_{i-1}, p_i, s'_{i-1})|. \tag{2}$$

If $\overline{w'_i}$ is west of s'_{i-1}, then by a convex path bound, we get

$$|\nearrow\!\!\supset_{\overline{w'_i},s'_{i-1}}| \leq |P_i| + |\text{Arc}(C'_{i-1}, p_i, s'_{i-1})|. \tag{3}$$

On the other hand, inequality 3 still holds when $\overline{w'_i}$ is east of s'_{i-1} since $|\nearrow\!\!\supset_{\overline{w'_i},s'_{i-1}}| = 0$. Finally,

$$\begin{aligned} |P_{i-1}| + |\text{Arc}(C'_{i-1}, p_{i-1}, p_i)| &= |\nearrow\!\!\supset_{\overline{w'_{i-1}},s'_{i-1}}| - |\text{Arc}(C'_{i-1}, p_i, s'_{i-1})| - |P_i| + |P_i| && \text{by 2} \\ &\leq |\nearrow\!\!\supset_{\overline{w'_{i-1}},s'_{i-1}}| - |\nearrow\!\!\supset_{\overline{w'_i},s'_{i-1}}| + |P_i| && \text{by 3} \\ &\leq |\nearrow\!\!\supset_{\overline{w'_{i-1}},\overline{w'_i}}| + |P_i| && \text{by Remark 11.} \end{aligned}$$

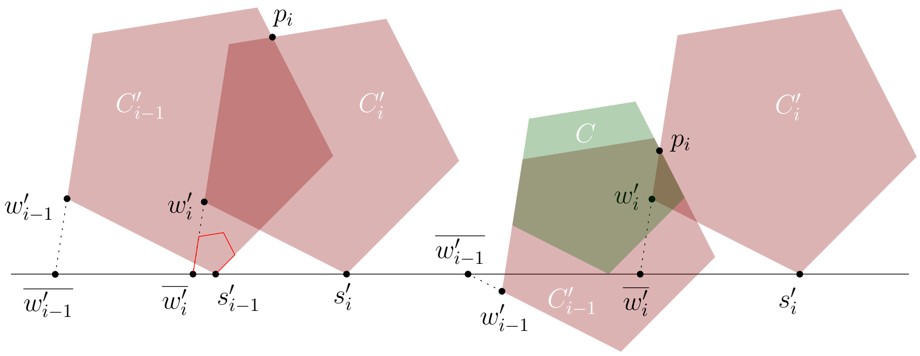


Figure 6 Left: Case 1 when $x(\overline{w'_i}) < x(s'_{i-1})$. The path $\diagup\!\!\sqsupset_{\overline{w'_i},s'_{i-1}}$ is red. Right: Case 2 reduces to case 1 by defining n-gon C in green.

Case 2: p_{i-1} is above st and only s'_i lies on st. We will reduce this case to Case 1. See Figure 6 First, define a new n-gon C with $w := \mathrm{West}(C)$ such that $\overline{w} = \overline{w'_{i-1}}$, and $\mathrm{South}(C), s, t$ are collinear, and $p_i \in \partial C$. By a convex path bound, we have

$$|P_{i-1}| + |\mathrm{Arc}(C'_{i-1}, p_{i-1}, p_i)| \leq |\overline{w'_{i-1}}w| + |\mathrm{Arc}(C, w, p_i)| \tag{4}$$

Lastly, we proceed with the argument in Case 1, replacing C'_{i-1} with C in order to obtain

$$|\overline{w'_{i-1}}w| + |\mathrm{Arc}(C, w, p_i)| \leq |P_i| + |\diagup\!\!\sqsupset_{\overline{w'_{i-1}},\overline{w'_i}}| \tag{5}$$

Combining (4) with (5) yields (1).

Case 3: p_{i-1} is above st and s'_i lies below st. In this case, $p_i = w'_i$, meaning $P_i = \overline{w'_i}p_i$. Let point p be collinear with $\overline{w'_{i-1}}, w'_{i-1}$ such that $y(p) = y(p_i)$. Then by a convex path bound, we have

$$|P_{i-1}| + |\mathrm{Arc}(C'_{i-1}, p_{i-1}, p_i)| \leq |\overline{w'_{i-1}}p| + |\diagup\!\!\sqsupset_{p,p_i}|$$

Finally, equation (1) follows since the paths $\overline{w'_{i-1}}p$ and $\diagup\!\!\sqsupset_{p,p_i}$ are translates of P_i and $\diagup\!\!\sqsupset_{\overline{w'_{i-1}},\overline{w'_i}}$ respectively.

Case 4: p_{i-1} is below st. In this case, $w'_{i-1} = p_{i-1}$, hence $P_{i-1} = \overline{w'_{i-1}}w'_{i-1}$. Define point p be the intersection of C'_{i-1} closest to s. We will prove the claim, illustrated in Figure 8.

▷ Claim. P_{i-1} is a sub-path of $\diagdown\!\!\sqsupset_{\overline{w'_{i-1}},p}$.

Fix m such that $\bigcirc_n^m := \mathrm{North}(\bigcirc_n)$. Then for $0 \leq j \leq m-2$ let $\hat{C}_j$ be the n-gon such that $\hat{C}_j^{m-j} = p$ and $\overline{\mathrm{West}(\hat{C}_j)} = \overline{w'_{i-1}}$. Note that $\hat{C}_0$ is exactly the n-gon corresponding to $\diagdown\!\!\sqsupset_{\overline{w'_{i-1}},p}$. In addition, p was defined to be collinear with $C'^1_{i-1}C'^2_{i-1}$, therefore $w'_{i-1} = \mathrm{West}(\hat{C}_{m-2})$. Then, for $1 \leq j \leq m-2$, the n-gons $\hat{C}_{j-1}, \hat{C}_j$ are related by a homothety with center c_j lying on the intersection of extended segments $\hat{C}_j^1\hat{C}_j^n$ and $\hat{C}_j^{m-j}\hat{C}_j^{m-j+1}$. By construction, we have $\hat{C}_j^{m-j} = \hat{C}_{j-1}^{m-j+1}$, therefore by symmetry (reflection about the extended line $c_j\mathrm{Center}(\hat{C}_j)$), we also have $\hat{C}_j^1 = \hat{C}_{j-1}^n$. Then $\mathrm{West}(\hat{C}_j)$ is on the snail path $\diagdown\!\!\sqsupset_{\overline{w'_{i-1}},p}$ if and only if $j < 2$. In particular, the claim holds when $m - 2 < 2$, which is the case for $n \in \{5, 6, 7, 8\}$ since in general $\lfloor \frac{n}{4} \rfloor \leq m - 1 \leq \lceil \frac{n}{4} \rceil$.

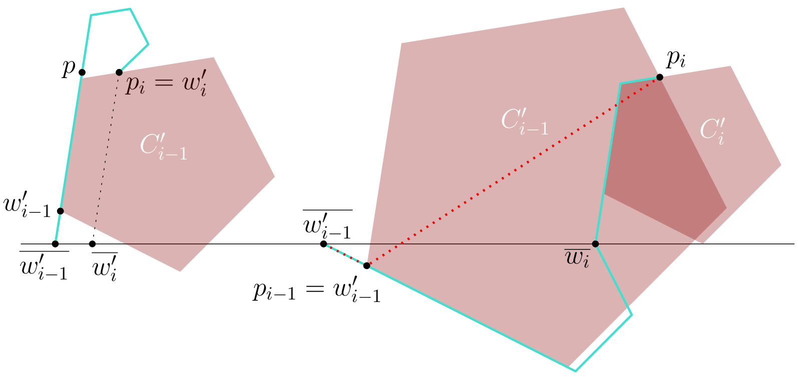


Figure 7 Left: In Case 3, the blue path $\overline{w'_{i-1}}X + \diagup\!\!\!\sqsupset_{X,p_i}$ is longer than the dotted red path $P_{i-1} + \mathrm{Arc}(C'_{i-1}, p_{i-1}, p_i)$. Right: In Case 4: The blue path $P_{i-1} + p_{i-1}p_i$ is longer than the dotted red path $\diagdown\!\!\!\sqsupset_{\overline{w'_{i-1}},\overline{w'_i}} + P_i$.

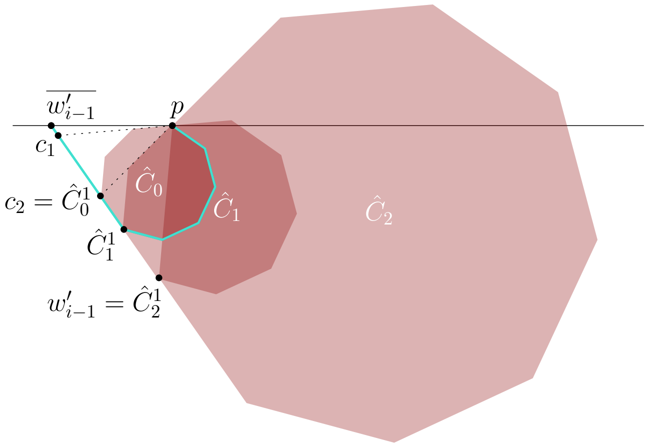


Figure 8 Case 4 claim. Here, $n = 9$, $m = 4$, and therefore $C'_{i-1} = \hat{C}^1_{m-2}$ is not on the blue path $\diagdown\!\!\!\sqsupset_{\overline{w'_{i-1}},p}$. If instead $n \in \{5, 6, 7, 8\}$, then $m \in \{2, 3\}$.

Then by Lemma 8, $x(p) \leq x(\overline{w'_i})$, therefore P_{i-1} is a also sub-path of $\diagdown\!\!\!\sqsupset_{\overline{w'_{i-1}},w'_i}$. Inequality (1) follows since the paths $P_{i-1}+p_{i-1}p_i$ and $\diagdown\!\!\!\sqsupset_{\overline{w'_{i-1}},\overline{w'_i}}+P_i$ have the same endpoints, concluding this case.

Finally, we have $|P_k| + |p_k t| \leq \max(|\diagup\!\!\!\sqsupset_{\overline{w'_k},t}|, |\diagdown\!\!\!\sqsupset_{\overline{w'_k},t}|)$ by a convex path bound. Note that for $i = 1$, the edge sp_1 classifies as Case 4. ◀

Putting this all together, we can now prove Theorem 3.

Proof. Consider the path $s = p_0, ..., p_{k+1} = t$ from Algorithm 1 and apply Lemma 14:

$$\begin{aligned}\sum_{i=1}^{k+1} |p_{i-1}p_i| &\leq (\sum_{i=1}^{k} |P_i| + \Psi(5)|\overline{w'_{i-1}w'_i}| - |P_{i-1}|) + (\Psi(5)|\overline{w'_k}t| - |P_k|)\\ &= \Psi(5)|\overline{w'_0}t| - |P_0| = \Psi(5)|st|.\end{aligned}$$

Since Algorithm 1 is a $O(\log(|V(G)|))$-memory local routing algorithm, then $\Psi(5)$ is an upper bound on the routing ratio of $\bigcirc_5$-Delaunay graphs. ◀

Now we present a lemma showing that our analysis is optimal for Algorithm 1.

▶ **Lemma 15.** *Let $n \in \{5, 6, 7, 8\}$. For any $\epsilon > 0$, there is a $\bigcirc_n$-Delaunay graph for which Algorithm 1 has a routing ratio of at least $\Psi(n) - \epsilon$.*

Proof. For simplicity, we present S not in general position, however it is possible to perturb the position of each vertex to force the $\bigcirc_n$-Delaunay graph of S to have the desired edges while being in general position. See Figure 9. Let C be an n-gon, and define σ such that $C^\sigma = \text{South}(C)$. Then let L be a horizontal line above and arbitrarily close to C^σ. Define $t := L \cap C^{\sigma-1}C^\sigma$ and $v_j := C^j$ for $j \in \{2, ..., \sigma - 1\}$. Let v_0 be on $C^\sigma C^{\sigma+1}$ below L. Let v_1 be on C^1C^2 arbitrarily close to C^1. Then let $\hat{C}$ be an n-gon such that $\text{East}(\hat{C}) = v_0$ and $p_1 \in \partial\hat{C}$. Finally, s is the leftmost intersection of L and $\partial\hat{C}$. Let $S := \{s, t, v_0, v_1, ..., v_{\sigma-1}\}$. While there are several valid $\bigcirc_n$-Delaunay graphs of S, we will choose to route on the graph with edges $sv_0, sv_1, v_{j-1}v_j, v_1v_0, v_jv_0, v_0t, v_{\sigma-1}t$ for $j \in \{2, ..., \sigma - 1\}$. When Algorithm 1 routes from s to t in the $\bigcirc_n$-Delaunay graph of S, then each step of case b is sub-case b1, therefore the path is $s, v_1, v_2, ..., v_{\sigma-1}, t$. By construction, this path is arbitrarily close to the snail path $\diagup\!\!\!\supset_{s,t}$. Since $\Psi(n)|st|$ represents the maximum length of the snail path $\diagup\!\!\!\supset_{s,t}$ over all orientations of $\bigcirc_n$, then the routing ratio of Algorithm 1 for this graph is at least $\Psi(n) - \epsilon$. ◀

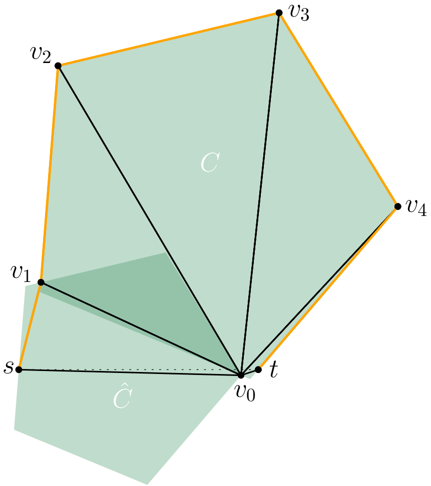


Figure 9 Worst-case point-set construction causes Algorithm 1 to choose the orange path which is arbitrarily close to $|\diagup\!\!\!\supset_{s,t}|$ since each step of case b is sub-case b1. Here $n = \sigma = 5$. For more details, see proof of Lemma 15.

3.2 Extending Algorithm 1 to graphs where the convex hull is not present

In order to extend Algorithm 1 to graphs where the convex hull is not present, we add dummy vertices to the graph G to ensure the entire set S is triangulated. In particular, define the set D of dummy vertices as follows:

Let C be the scaled translate of $\bigcirc_n$ with $\text{Center}(C) = s$ and $|\text{West}(C)\text{Center}(C)| = 10|st|$. Then let $\text{Rot}(C)$ denote the rotated n-gon obtained by rotating C about $\text{Center}(C)$ by π radians (or by $\pi - \epsilon$ radians for $\epsilon > 0$ to satisfy the general position assumption). Lastly, let D be the corners of $\text{Rot}(C)$ that are in the unbounded region of the $\bigcirc_n$-Delaunay graph of S.

If Algorithm 1 is used in the $\bigcirc_n$-Delaunay graph of $S \cup D$, then the path from s to t will have length at most $c|st|$ where c is given in Lemma 16. In all cases, $c < 10$, so for all $i \in \{0, ..., k+1\}$, we have that $p_i \notin D$. The choice of D ensures that all p_i are not incident to the unbounded region.

When routing in the $\bigcirc_n$-Delaunay graph of S, Algorithm 1 can be modified to simulate the edges of the $\bigcirc_n$-Delaunay graph of $S \cup D$. Firstly, it can be verified locally whether the current vertex p_i is incident to the unbounded region. Indeed, p_i is in the unbounded region if and only if there exists a $j \in \{1, ..., n\}$ and an n-gon C such that $C^j = p_i$ and no edges incident to p_i intersect C. Note that the size of C is irrelevant. Then if p_i is incident with the unbounded region, it can be verified whether p_i has neighbours in D in the $\bigcirc_n$-Delaunay graph of $S \cup D$ since the algorithm stores the location of s and can calculate the distance $|st|$.

3.3 Extending our result to hexagons, septagons and octagons

So far, we have shown that Algorithm 1 has a routing ratio of at most $\Psi(5) \approx 4.640$ for any $\bigcirc_5$-Delaunay graph. To extend our result to n-gons with $6 \leq n \leq 8$, then Lemma 12 needs to be generalized to Lemma 16. Additionally, Lemma 14 is trivially generalized by replacing each occurrence of $\Psi(5)$ with $\Psi(n)$ for the corresponding n-gon.

▶ **Lemma 16.** *Let $a, b \in \mathbb{R}^2$ with $y(a) = y(b)$ and $x(a) < x(b)$. Then* $\max(|{\diagup\!\!\sqsupset}_{a,b}|, |{\diagdown\!\!\sqsupset}_{a,b}|) \leq \Psi(n)|ab|$ *for $n \in \{5, 6, 7, 8\}$.*

Proof. Let $n \in \{5, 6, 7\}$ and let C be the n-gon corresponding to ${\diagup\!\!\sqsupset}_{a,b}$. Let $\theta := \angle\mathrm{West}(C)ab$, and for now assume that $\mathrm{South}(C) = C^n$. This means that $\pi/5 \leq \theta \leq \pi/2$. Then by the law of sines, $|\mathrm{West}(C)\mathrm{South}(C)| = \frac{|ab|\sin(\theta)}{\sin(2\pi/n)}$ and $|a\mathrm{West}(C)| = \frac{|ab|\sin(2\pi/n+\theta)}{\sin(2\pi/n)}$. Since $|{\diagup\!\!\sqsupset}_{a,b}| = |a\mathrm{West}(C)| + (n-1)|\mathrm{West}(C)\mathrm{South}(C)|$, then we have

$$\frac{|{\diagup\!\!\sqsupset}_{a,b}|}{|ab|} = \frac{\sin(2\pi/n+\theta)}{\sin(2\pi/n)} + (n-1)\frac{\sin(\theta)}{\sin(2\pi/n)}. \tag{6}$$

Focusing on the numerators, we have

$$\begin{aligned}\frac{d}{d\theta}\Big(\sin(2\pi/n+\theta) + (n-1)\sin(\theta)\Big) &= \cos(2\pi/n+\theta) + (n-1)\cos(\theta)\\ &= (\cos(2\pi/n)\cos(\theta) - \sin(2\pi/n)\sin(\theta)) + (n-1)\cos(\theta)\\ &= (\cos(2\pi/n) + n - 1)\cos(\theta) - \sin(2\pi/n)\sin(\theta).\end{aligned}$$

From there, we get that the critical value of θ is

$$\theta^* = \arctan\left(\frac{\cos(2\pi/n) + n - 1}{\sin(2\pi/n)}\right).$$

Therefore the maximum value is

$$\frac{\sin\left(2\pi/n + \theta^*\right) + (n-1)\sin\left(\theta^*\right)}{\sin(2\pi/n)}.$$

When n is 5, 6 or 7, the maximum value of (6) is approximately 4.640, 6.429, or 8.531 respectively. It is straightforward to verify that the claim still holds when $\mathrm{South}(C) \neq C^n$.

When $n = 8$, the analysis is slightly different since we can not have $\mathrm{South}(C) = C^n$. As before, let C be the 8-gon corresponding to ${\diagup\!\!\sqsupset}_{a,b}$, and let $\theta := \angle\mathrm{West}(C)ab$. Then necessarily, we have $\mathrm{South}(C) = C^7$, meaning that $\angle bC^1a = \frac{3\pi}{8}$. By the law of sines, $|C^1C^7| = \frac{|ab|\sin(\theta)}{\sin(3\pi/8)}$ and $|aC^1| = \frac{|ab|\sin(3\pi/8+\theta)}{\sin(3\pi/8)}$. Also, $|\mathrm{Arc}(C, C^7, C^1)|\cos(\pi/8) = |C^1C^7|$. Since $|{\diagup\!\!\sqsupset}_{a,b}| = |aC^1| + 3|\mathrm{Arc}(C, C^7, C^1)|$, then we have

$$\frac{|\nearrow_{a,b}|}{|ab|} = \frac{\sin(3\pi/8+\theta)}{\sin(3\pi/8)} + 3\frac{\sin(\theta)}{\sin(3\pi/8)\cos(\pi/8)} \text{ where } \pi/4 \le \theta \le \pi/2,$$

which reaches a maximum of around 4.054 by a similar analysis to above. ◀

4 Routing Ratio Lower Bound for $\bigcirc_5$-Delaunay graphs

▶ **Lemma 17.** *Every local routing algorithm for $\bigcirc_5$-Delaunay graphs must have a routing ratio of at least* 1.995 *for any* $\epsilon > 0$.

Proof. Consider the construction shown in Figure 10.

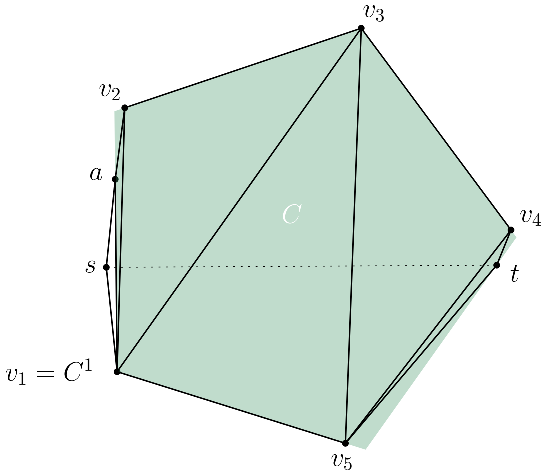


Figure 10 Point set construction obtaining a routing ratio lower bound of approximately 1.995.

Let $\delta > 0$ and let C be a pentagon with $|C^1C^2| = 1 + \delta$. Define $v_1 := C^1$, $v_2 \in C^2C^3$ arbitrarily close to C^2, $v_3 := C^3$, $v_4 \in C^3C^4$ arbitrarily close to C^4, and $v_5 \in C^5C^1$ arbitrarily close to C^5. Finally place point a on C^1C^2 such that $|C^1a| = 1$. Place s outside C equidistant from v_1, a, arbitrarily close to C^1C^2. Define the point $t \in C^4C^5$ on the line perpendicular to C^1C^2 through s. The point set is $S := \{s, t, a, v_1, v_2, v_3, v_4, v_5\}$. While S admits many different triangulations, we choose the triangulation from Figure 10. The edges are $sa, sv_1, av_1, av_2, v_1v_2, v_1v_3, v_1v_5, v_2v_3, v_3v_4, v_3v_5, v_4v_5, v_4t, v_5t$. In particular, the neighbourhood of s is $\{a, v_1\}$, however the shortest path from a to t is approximately along the boundary ∂C. We will analyze the routing ratio of any algorithm that chooses to go to a. Then, any algorithm that chooses v_1 first will perform poorly on a symmetric graph reflected about the line through st.

We consider the clockwise and counterclockwise arcs from point a to point t. The counterclockwise arc has length

$$\begin{aligned}|\mathrm{Arc}(C_1, t, a)| &= |av_1| + |v_1C^5| + |C^5t| \\ &= 1 + (1+\delta) + \frac{\frac{1}{2} + (1+\delta)\sin 18}{\sin 54}.\end{aligned}$$

On the other hand, the clockwise arc from a to t has length

$$\begin{aligned}|\mathrm{Arc}(C_1, a, t)| &= |aC^2| + |C^2C^3| + |C^3C^4| + |C^4t| \\ &= \delta + (1+\delta) + (1+\delta) + (1 + \delta - \frac{\frac{1}{2} + (1+\delta)\sin 18}{\sin 54}).\end{aligned}$$

Finally, we have $|st| = |v_1C^5|\cos 18 + |C^5t|\cos 54 = (1+\delta)\cos 18 + \frac{\frac{1}{2}+(1+\delta)\sin 18}{\tan 54}$. This means that the routing ratio is at least

$$\frac{|sa| + |\text{ShortestPath}(a,t)|}{|st|} = \frac{\frac{1}{2} + \min(|\text{Arc}(C_1,a,t)|, |\text{Arc}(C_1,t,a)|)}{|st|}$$

which reaches approximately 1.995 when $\delta = 0.447$. ◀

5 Spanning Ratio Lower Bound for $\bigcirc_5$-Delaunay graphs

▶ **Lemma 18.** *For any $\epsilon > 0$, there exists a $\bigcirc_5$-Delaunay graph with spanning ratio of at least $\frac{5}{2+3\sin(\frac{\pi}{10})} - \epsilon$.*

Proof. Let $\epsilon > 0$. We will construct a point set for the $\bigcirc_5$-Delaunay graph shown in Figure 11. In particular, let the pentagon C have side length 1, then place a on C^1C^5 with $|C^1a| = \frac{1}{4}$ and b on C^3C^4 with $|C^3b| = \frac{1}{4}$. Next, place $v_1 \in C^1C^2$ arbitrarily close to C^1, $v_2 := C^2$, $v_3 \in C^2C^3$ arbitrarily close to C^3, $v_4 \in C^4C^5$ arbitrarily close to C^4, and $v_5 \in C^4C^5$ arbitrarily close to C^5. We define $S := \{a, b, v_1, v_2, v_3, v_4, v_5\}$ and consider the triangulation with edges $av_1, av_5, v_1v_2, v_1v_5, v_2v_3, v_2v_4, v_2v_5, v_3v_4, v_3b, v_4b, v_4v_5$. The shortest path from a to b is approximately the boundary ∂C. Therefore the length of the shortest path from a to b is approximately $\frac{5}{2}$, whereas $|ab| = 1 + \frac{3}{2}\sin(\frac{\pi}{10})$. This means that the spanning ratio of is approximately 1.708. ◀

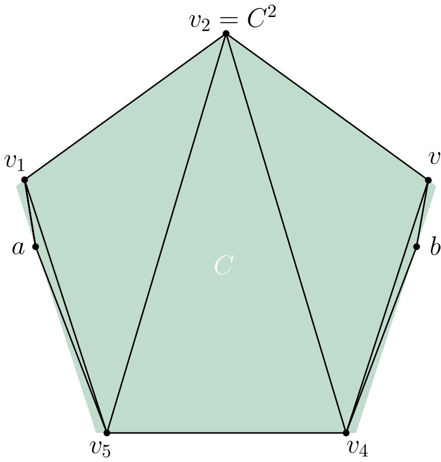


Figure 11 Point set construction obtaining a spanning ratio lower bound of approximately 1.708. The shortest path between a and b is arbitrarily close to the perimeter of the pentagon.

6 Conclusions

We have dramatically improved the upper bound on the spanning ratio when $n \in \{5, 7, 8\}$ and on the routing ratio when $n \in \{5, 6, 7, 8\}$. We also provided matching lower bounds for the paths output by our routing algorithm, showing that our analysis is tight. Furthermore, we prove that no routing algorithm for pentagon-Delaunay graphs can have a routing ratio less than 1.995. Finally, we show that the worst-case spanning ratio is at least 1.708 for the pentagon-Delaunay graph. We conclude with the following three open questions: (1) Can we improve the lower bound on the routing ratio for these graphs? (2) Can we provide a routing algorithm that attains this lower bound? (3) Can our approach be generalized for regular polygons with more than 8 sides?

References

1 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Vincent Despré, Darryl Hill, and Michiel Smid. Improved routing on the Delaunay triangulation. *Discret. Comput. Geom.*, 70(3):495–549, 2023. doi:10.1007/s00454-023-00499-9.

2 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Ljubomir Perkovic, and André van Renssen. Upper and lower bounds for online routing on Delaunay triangulations. *Discret. Comput. Geom.*, 58(2):482–504, 2017. doi:10.1007/s00454-016-9842-y.

3 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perkovic. Tight stretch factors for L_1- and L_∞-Delaunay triangulations. *Comput. Geom.*, 48(3):237–250, 2015. doi:10.1016/j.comgeo.2014.10.005.

4 Prosenjit Bose, Paz Carmi, Sébastien Collette, and Michiel H. M. Smid. On the stretch factor of convex Delaunay graphs. *J. Comput. Geom.*, 1(1):41–56, 2010. doi:10.20382/jocg.v1i1a4.

5 Prosenjit Bose, Jean-Lou De Carufel, Stephane Durocher, and Perouz Taslakian. Competitive online routing on Delaunay triangulations. *Int. J. Comput. Geom. Appl.*, 27(4):241–254, 2017. doi:10.1142/S0218195917500066.

6 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM J. Comput.*, 44(6):1626–1649, 2015. doi:10.1137/140988103.

7 Prosenjit Bose and Pat Morin. Competitive online routing in geometric graphs. *Theor. Comput. Sci.*, 324(2-3):273–288, 2004. doi:10.1016/j.tcs.2004.05.019.

8 Paul Chew. There is a planar graph almost as good as the complete graph. In *SCG*, pages 169–177. ACM, 1986. doi:10.1145/10515.10534.

9 Paul Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, 1989. doi:10.1016/0022-0000(89)90044-5.

10 Lihong Ma. *Bisectors and Voronoi Diagrams for Convex Distance Functions*, volume 267 of *Informatik-Berichte*. FernUniversität in Hagen, Hagen, 2000. Zugl.: Dissertation, FernUniversität in Hagen, 2000. URL: https://ub-deposit.fernuni-hagen.de/receive/mir_mods_00000857.

11 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

12 Ljubomir Perkovic, Michael Dennis, and Duru Türkoglu. The stretch factor of hexagon-Delaunay triangulations. *J. Comput. Geom.*, 12(2):86–125, 2021. doi:10.20382/jocg.v12i2a5.

13 Ge Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM J. Comput.*, 42(4):1620–1659, 2013. doi:10.1137/110832458.

14 Ge Xia and Liang Zhang. Toward the tight bound of the stretch factor of Delaunay triangulations. In *CCCG*, 2011. URL: http://www.cccg.ca/proceedings/2011/papers/paper57.pdf.

On the Spanning and Routing Ratios of the Yao-Four Graph

Prosenjit Bose
Carleton University, Ottawa, Canada

Darryl Hill
Carleton University, Ottawa, Canada

Michiel Smid
Carleton University, Ottawa, Canada

Tyler Tuttle
Carleton University, Ottawa, Canada

Abstract

The Yao graph is a geometric spanner that was independently introduced by Yao [SIAM J. Comput., 1982] and Flinchbaugh and Jones [SIAM J. Algebr. Discret. Appl., 1981]. We prove that for any two vertices of the undirected version of the Yao graph with four cones, there is a path between them with length at most $13 + 5/\sqrt{2} \approx 16.54$ times the Euclidean distance between the vertices, improving the previous best bound of approximately 54.62. We also present an online routing algorithm for the directed Yao graph with four cones that constructs a path between any two vertices with length at most $17 + 9/\sqrt{2} \approx 23.36$ times the Euclidean distance between the vertices. This is the first routing algorithm for a directed Yao graph with fewer than six cones. The algorithm uses knowledge of the coordinates of the current vertex, the (up to) four neighbours of the current vertex, and the destination vertex to make a routing decision. It also uses one additional bit of memory. We show how to dispense with this single bit at the cost of increasing the length of the path to $\sqrt{331 + 154\sqrt{2}} \approx 23.43$ times the Euclidean distance between the vertices.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Yao graph, online routing, geometric spanners

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.15

Acknowledgements We would like to thank an anonymous reviewer for suggestions to improve the presentation and for pointing out an error in our original proof of Lemma 7.

1 Background

Online routing is the problem of constructing a path in a graph from some current vertex to a given destination vertex, without knowing the entire graph ahead of time. The path must be constructed one vertex at a time. A *routing algorithm* computes, given some vertex, the next vertex on the path.

The information available to a routing algorithm is of great importance. We say that a routing algorithm is *local* if the only information available to it is knowledge of the current vertex, the immediate neighbours of the current vertex, the destination vertex, plus a constant amount of extra information.

In this paper we will consider routing algorithms for a specific type of geometric graph called a Yao graph. A geometric graph is a graph whose vertex set is a set of points in the plane, and whose edges are weighted by the Euclidean distance between their endpoints. We will consider both directed and undirected graphs. Since the vertices of a geometric graph are points, we will assume that the routing algorithm has access to their coordinates.

Let u and v be two points in the plane. Define $d_x(u,v)$ and $d_y(u,v)$ to be the horizontal and vertical distances between u and v. In this paper we will make use of three different distance functions, or metrics, on $\mathbf{R}^2$. They are the L_1 metric, the L_2 (or Euclidean) metric, and the L_∞ metric.

$$\begin{aligned} \|uv\|_1 &= d_x(u,v) + d_y(u,v) \\ \|uv\|_2 &= \sqrt{d_x(u,v)^2 + d_y(u,v)^2} \\ \|uv\|_\infty &= \max\{d_x(u,v), d_y(u,v)\} \end{aligned} \tag{1}$$

All of the results will be in terms of the standard Euclidean distance, but the analysis will make use of the other metrics.

Let $G = (V,E)$ be a geometric graph, directed or undirected. Let $d_G(u,v)$ denote the length of the shortest path from u to v in G. For a real number t, we say that G is a t-spanner if $d_G(u,v) \leq t\|uv\|_2$ for all u and v in V. The *spanning ratio* (also called the stretch factor) of G is the minimum such t.

Let $\mathcal{A}$ be a routing algorithm, and let $d_G^{\mathcal{A}}(u,v)$ denote the length of the path from u to v in G constructed by algorithm $\mathcal{A}$. We say that $\mathcal{A}$ is t-competitive if $d_G^{\mathcal{A}}(u,v) \leq t\|uv\|_2$ for all u and v in G. The *routing ratio* of $\mathcal{A}$ is the smallest such t. If $\mathcal{A}$ is t-competitive for all graphs in some class $\mathcal{G}$, then we say that $\mathcal{A}$ is t-competitive on $\mathcal{G}$. The precise definition of an online routing algorithm will be given in Section 1.2.

1.1 Yao graphs

The results of this paper are about the spanning and routing ratios of a specific class of geometric graph known as Yao graphs. We begin by defining these graphs.

Fix an integer $k \geq 3$. Let R_i be the ray emanating from the origin making an angle of $2\pi i/k$ with the positive x-axis[1]. The region between two consecutive rays is called a cone. Specifically, let C_i be the region between R_i and R_{i+1}, including R_i but not R_{i+1}. The k cones C_0 through C_{k-1} partition the plane (minus the origin) into k regions. Let $C_i(p)$ and $R_i(p)$ be C_i and R_i translated so that the rays emanate from p rather than the origin.

Given a set P of n points in the plane, we construct a directed graph in the following way. For each point p in P, we add at most k edges to the graph, one edge per cone $C_i(p)$ of p. Let q be the point in $P \cap C_i(p)$ that minimizes the distance $\|pq\|_2$. Add a directed edge from p to q. See Figure 1 for an example. Repeating this for every point and every cone adds at most kn edges to the graph. The resulting graph is called the (directed) Yao-k graph of P, denoted $\vec{Y}_k(P)$ or simply $\vec{Y}_k$ if the set P is clear from context. The undirected Yao-k graph $Y_k(P)$ (or Y_k if the set P is clear) is obtained by forgetting the direction of each edge of $\vec{Y}_k(P)$.

Notice that the vertices of a directed Yao-k graph have outdegree bounded by k. The vertices of an undirected Yao-k graph can have unbounded degree.

The Yao graph was first defined in the early 1980s independently by Flinchbaugh and Jones [1] and Yao [13]. Althöfer et al. first proved that Yao graphs are spanners in 1993 [6]. Specifically, they showed that for any set P of points and any $t \geq 1$, there is an integer k such that $Y_k(P)$ is a t-spanner of P.

Later work found specific bounds for spanning ratios of Yao graphs. Bose et al. showed that the spanning ratio of $Y_k(P)$ is at most $1/(\cos\theta - \sin\theta)$ for all $k \geq 9$ [11], where $\theta = 2\pi/k$. This was later improved to $1/(1 - 2\sin(\theta/2))$ for all $k \geq 7$ [12]. Finally, Barba et al. showed that the spanning ratio is at most $1/(1 - 2\sin(3\theta/8))$ for odd values of $k \geq 5$ [2].

[1] Angles are measured counterclockwise.

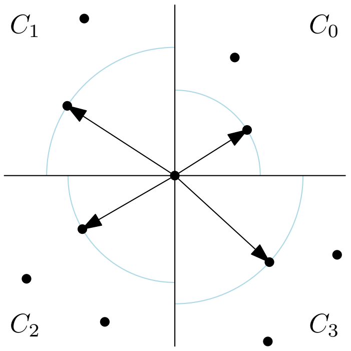


Figure 1 The four cones used to define the $\vec{Y}_4$ graph.

The techniques used to bound the spanning ratio of Yao graphs with many cones do not directly translate to Yao graphs with very few cones. For these, more specific arguments are required. For Y_6, the first published bound was by Damian and Raudonis, who showed that the spanning ratio is at most 17.64 [9]. This was later improved by Barba et al. to 5.8 [2].

In the same paper that gives a bound of $1/(1-2\sin(3\theta/8))$ on the spanning ratio of Y_5, Barba et al. [2] give a more precise argument that the spanning ratio of Y_5 is at most $2+\sqrt{3}$. For Y_4, the first bound was by Bose et al. [12], who showed that the spanning ratio is at most 662. This was improved by Damian and Nelavalli to 54.62 [10].

For fewer than four cones Yao graphs are not necessarily spanners [5]. In fact, the Y_1 graph is equivalent to the (directed) nearest-neighbour graph, which may not even be connected. The Y_2 and Y_3 graphs are connected, but the directed versions are not necessarily strongly connected. These results are summarized in Table 1.

For more than six cones, the proofs that Yao graphs are spanners give a routing algorithm called cone routing: always move to the neighbour in the cone that contains the destination. This gives a routing ratio equal to the spanning ratio. For $\vec{Y}_6$, a local routing algorithm is known with a routing ratio of $1+38/\sqrt{3} \approx 22.94$ [7]. For four or five cones, however, no local routing algorithms are known.

Theta graphs [4, 8] are a class of graphs that are similar to Yao graphs. The construction begins the same way, by partitioning space into cones, but instead of adding an edge from a vertex to its nearest neighbour in each cone, the vertices in each cone are projected onto the bisector of the cone and an edge is added to whichever vertex has the closest projection. Like Yao graphs, Theta graphs are spanners, and routing algorithms for Theta graphs have been studied. In this paper we will present an online routing algorithm for $\vec{Y}_4(P)$ that is a modification of the best-known routing algorithm for $\vec{\Theta}_4(P)$ [3]. Although the algorithm is similar, some care must be taken since every edge of a Theta graph defines an empty triangle, whereas every edge of a Yao graph defines an empty sector of a circle. More importantly, our analysis of the algorithm is novel and we believe it is much simpler than the analysis for the $\vec{\Theta}_4(P)$ routing algorithm.

1.2 Local routing

Let $G = (V, E)$ be a graph. An online routing algorithm on G can be modelled as a function $f : V \times V \times P(V) \times \{0,1\}^* \to V \times \{0,1\}^*$, where $P(V)$ is the power set of V.

The first two parameters of f are the current vertex u and the target vertex v. The third parameter is the set of vertices we can use to make a routing decision. We will focus on *local* routing, where this parameter is the set of neighbours of u. The fourth parameter is a finite bitstring called the *header*, which a routing algorithm can use to store arbitrary information as it constructs a path.

Table 1 Lower and upper bounds for spanning ratios of Yao graphs ($\theta = 2\pi/k$). Our improved upper bound for $k = 4$ is highlighted in bold. In the bottom four rows, m is a positive integer.

Number of cones k	Lower bound	Upper bound
3 or fewer	∞	∞
4	Open	**16.54**
5	2.87	3.74
6	2	5.8
$4m+2$	$1+2\sin(\theta/2)$	$\dfrac{1}{1-2\sin(\theta/2)}$
$4m+3$	$1+2\sin\left(\frac{3\theta}{8}\right)\left(1+2\left(\sin\left(\frac{13\theta}{16}\right)+\sin\left(\frac{19\theta}{16}\right)\right)\frac{\sin(\theta/16)}{\sin(2\theta)}\right)$	$\dfrac{1}{1-2\sin(3\theta/8)}$
$4m+4$	$1+2\sin(\theta/2)(1+\tan(\theta/2))$	$\dfrac{1}{1-2\sin(\theta/2)}$
$4m+5$	$1+2\sin(3\theta/8)+4\sin(5\theta/16)\sin(3\theta/8)$	$\dfrac{1}{1-2\sin(3\theta/8)}$

A routing algorithm must take these four parameters and decide which neighbour of u should come next on the path, and potentially modify the header in some way. These are the two outputs of the function f. A path from a vertex u to a vertex v is constructed in the following way. Let $u_0 = u$ and h_0 be some initial header that the routing algorithm is allowed to compute before constructing the path. Then we get a sequence of vertices and headers defined by $(u_{i+1}, h_{i+1}) = f(u_i, t, N(u_i), h_i)$. If $u_i = v$ for some i, we stop as the routing algorithm has successfully found a path from u to v.

If, for any two vertices u and v in G, a routing algorithm $\mathcal{A}$ constructs a path from u to v, then we say that $\mathcal{A}$ guarantees delivery on the graph G. Let $d_G^{\mathcal{A}}(u, v)$ denote the Euclidean length of the path from u to v in G constructed by algorithm $\mathcal{A}$. We say that $\mathcal{A}$ is t-competitive if $d_G^{\mathcal{A}}(u, v) \leq t\|uv\|_2$. The *routing ratio* of $\mathcal{A}$ on G is the smallest t such that $\mathcal{A}$ is t-competitive.

If h_i is the empty bitstring for all i, then we say that $\mathcal{A}$ is *memoryless*. If $h_i = h_0$ for all i, then we say that $\mathcal{A}$ uses a static header. That is, the header is computed before the routing algorithm begins and never modified. Otherwise we say that $\mathcal{A}$ uses a dynamic header. The *memory usage* of a routing algorithm is the maximum length of h_i at any point during construction of a path from u to v, over all pairs of vertices in G.

2 The greedy-sweep algorithm

Let P be a finite set of points in the plane. To avoid tedious case analysis, we will make a general position assumption that no two points have the same x or y coordinate, and that no two points lie on a line with slope $+1$ or -1. Let s and t be two points of P. In this section we will describe the routing algorithm for constructing a path from s to t in the directed graph $\vec{Y}_4(P)$. First, we give some definitions that will be needed to describe and analyze the algorithm.

Let p be any point in the plane. For another point q, define W_{pq} to be the quarter-circle in $C_i(p)$ that is centred at p with q on its boundary contained in $C_i(p)$. Define the *diagonals* of p to be the lines through p with slope $+1$ and -1. Denote the diagonals of p by $d_+(p)$ and $d_-(p)$. The first step is to choose one of the two diagonals of t. Given the positions of s and t, choose whichever diagonal is closer to s. This can be determined by checking which cone of t contains s. If s is in $C_0(t)$ or in $C_2(t)$, then choose $d_+(t)$. Otherwise, choose $d_-(t)$. Let d denote the chosen diagonal. Knowledge of this diagonal will be needed to make routing decisions. The choice of diagonal needs exactly one bit of memory to be remembered. Later we will see that this bit can be dispensed with, at the cost of a slightly higher routing ratio. Finally, define the *height* of a point p, denoted $h(p)$, to be the L_1 distance from p to d.

For any point p of P, exactly one of the cones of p has a nonempty, bounded intersection with d. We say that this cone of p *faces* d. The intersection of the halfplane of d that contains p and the cone forms a right triangle. Denote this triangle by $T(p, d)$.

Algorithm 1 Pseudocode for the greedy-sweep algorithm.

```
            ▷ The procedure GREEDY($p$, $t$, $d$) returns the neighbour of $p$ in the cone that
contains $t$, and the procedure SWEEP($p$, $t$, $d$) returns the neighbour of $p$ in the cone that
faces diagonal $d$, or null if such a neighbour does not exist.
procedure GREEDYSWEEP($p$, $t$, $d$)
    $q \leftarrow$ GREEDY($p, t, d$)
    $r \leftarrow$ SWEEP($p, t, d$)
    if $r =$ null or $\|pr\|_2 > h(p)$ then
        return $q$
    else
        return $r$
    end if
end procedure
```

To make a routing decision at a point p, we have a choice of up to four neighbours. We will determine where to go in the following way. Consider the triangle $T(p, d)$. If $T(p, d)$ is empty of points of P, then we say that it is *clean*, or that p is clean with respect to d. Given the information at p, it might not be possible to determine if $T(p, d)$ is clean or not. However, if r is the neighbour of p in the cone containing $T(p, d)$, and $\|pr\|_2 > h(p)$, then $T(p, d)$ *must* be clean. Also, if p does not have a neighbour in the cone that faces d, then $T(p, d)$ is clean. If either of these two conditions are met, then move from p to the neighbour of p in whichever cone contains t. This is called a *greedy step*. Otherwise, move from p to its neighbour in the cone that faces d. This is called a *sweeping step*. See Figure 2.

In two cones of t, a sweeping step and a greedy step are identical, in the sense that they both select the same neighbour of p. For example, if $d = d_-(t)$ and if p is in $C_0(t)$, then a greedy step and a sweeping step will both choose the neighbour of p in $C_2(p)$. In this case we will consider the move to be a sweeping step. This means that greedy steps are only possible in two cones of t. Which two cones will depend on whether $d = d_-(t)$ or $d = d_+(t)$.

All of the information necessary to make a routing decision can be determined solely from the coordinates of p, the neighbours of p, and t, as well as the slope of the diagonal d. See Algorithm 1 for a pseudocode description of the algorithm.

3 Analysis

We begin by stating a lemma that will be very useful in our analysis, relating the three different metrics under consideration.

▶ **Lemma 1.** *For any points u and v in the plane, the following chain of inequalities holds:*

$$\|uv\|_\infty \leq \|uv\|_2 \leq \|uv\|_1 \leq \sqrt{2}\|uv\|_2 \leq 2\|uv\|_\infty. \tag{2}$$

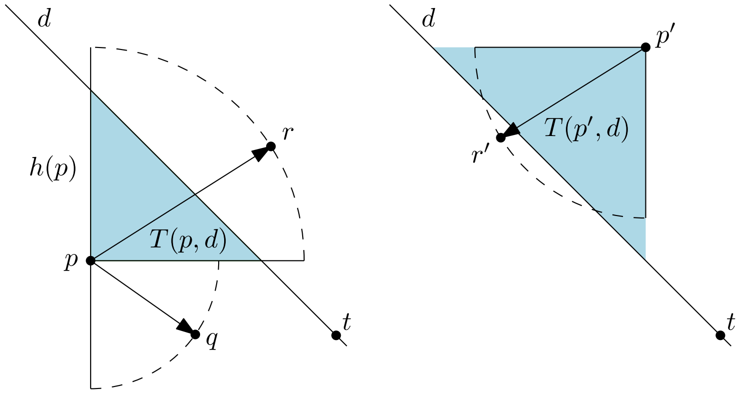


Figure 2 From p, a greedy step to q would be taken since $\|pr\|_2 > h(p)$. This implies that $T(p,d)$ is empty. From p' a sweeping step to r' would be taken since $\|p'r'\|_2 < h(p')$. There is not enough information at p' to determine if $T(p',d)$ is empty or not, since part of $T(p',d)$ is outside of the quarter circle defined by p' and r'.

Let P be a finite set of points in the plane, and let s and t be points in P. In this section we will analyze the path in $\vec{Y}_4(P)$ constructed by the greedy-sweep algorithm starting from s, with t as the destination. We begin the analysis by proving that the greedy-sweep algorithm guarantees delivery, by eventually reaching t. For the remainder of this section, we will assume without loss of generality that s is in $C_1(t)$, so $d = d_-(t)$, and that s is below d.

▶ **Lemma 2.** *Let s and t be different vertices of a $\vec{Y}_4$ graph, and consider the path constructed by the greedy-sweep algorithm starting at s with t as the destination. Let u and v be two consecutive vertices on this path. Then v is inside the square centred at t with u on its boundary, and so $\|ut\|_\infty > \|vt\|_\infty$.*

This lemma immediately implies that the algorithm terminates.

▶ **Corollary 3.** *The greedy-sweep algorithm guarantees delivery on the $\vec{Y}_4$ graph.*

To analyze the routing ratio of the greedy-sweep algorithm, we will consider greedy steps and sweeping steps separately. Recall that d is the diagonal fixed by the algorithm.

Consider an edge uv traversed while routing. Since no two points lie on a common diagonal by our general position assumption, the height change $h(v) - h(u)$ must be either positive or negative. Let D be the set of edges uv such that the height decreases from u to v, meaning $h(u) > h(v)$, and let I be the set of edges such that the height increases from u to v. Since $h(t) = 0$ because t lies on d, we know that the total decrease in height must equal the total increase in height plus the height of the initial point s. That is,

$$\sum_{uv \in D} \big(h(u) - h(v)\big) = h(s) + \sum_{uv \in I} \big(h(v) - h(u)\big). \tag{3}$$

The next two lemmas show that a sweeping step will always result in a height decrease that is at least proportional to the length of the edge, and that the length of a greedy edge is at least equal to the change in height.

▶ **Lemma 4.** *Let uv be an edge taken during a sweeping step. We have $h(u) - h(v) \geq (2 - \sqrt{2})\|uv\|_2$.*

Proof. We will consider two cases. See Figure 3. First, if uv does not cross d, then $h(v) = h(u) - \|uv\|_1$ and we have $h(u) - h(v) = \|uv\|_1 \geq \|uv\|_2$ by Lemma 1.

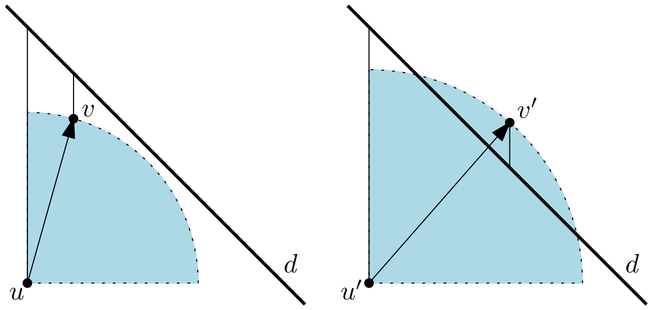


Figure 3 Illustration of Lemma 4. The edge uv does not cross d, and the edge $u'v'$ does. In both cases the height decreases when traversing the edge.

Suppose uv crosses d, and that v is in $C_i(u)$. Let θ be the angle made by the edge uv with the ray $R_i(u)$. We have $h(v) = \|uv\|_1 - h(u)$, so Suppose uv crosses d, and that uv is an i-edge. Let θ be the angle made by the edge uv with the ray $R_i(u)$. We have $h(v) = \|uv\|_1 - h(u)$, so

$$\begin{aligned} h(u) - h(v) &= 2h(u) - \|uv\|_1 \\ &\geq 2\|uv\|_2 - (\sin\theta + \cos\theta)\|uv\|_2 \\ &= (2 - (\sin\theta + \cos\theta))\|uv\|_2 \\ &\geq (2 - \sqrt{2})\|uv\|_2. \end{aligned}$$ ◀

▶ **Lemma 5.** *Let uv be an edge taken during a greedy step. We have $h(v) - h(u) \leq \|uv\|_2$.*

Proof. Assume without loss of generality that u is in $C_1(t)$ and that $d = d_-(t)$. If $h(v) < h(u)$, then the inequality is trivially satisfied. If $h(v) > h(u)$, then we have $h(v) - h(u) = d_y(u, v) \leq \|uv\|_2$. ◀

Now let S be the set of sweeping edges and G be the set of greedy edges. Lemma 4 implies $S \subseteq D$, so we must have $I \subseteq G$. In other words a sweeping edge will always decrease the height, so if an edge increases the height it must be a greedy edge.

▶ **Lemma 6.** *The total length of all the sweeping edges is at most $(1+\frac{1}{\sqrt{2}})(h(s)+\sum_{uv\in G}\|uv\|_2)$.*

Proof. The proof follows from Lemmas 4 and 5, and some simple manipulations:

$$\begin{aligned} \sum_{uv\in S}\|uv\|_2 &\leq \sum_{uv\in D}\frac{1}{2-\sqrt{2}}\big(h(u) - h(v)\big) \\ &= \frac{1}{2-\sqrt{2}}\sum_{uv\in D}\big(h(u) - h(v)\big) \\ &= \frac{1}{2-\sqrt{2}}\left(h(s) + \sum_{uv\in I}\big(h(v) - h(u)\big)\right) \\ &\leq \frac{1}{2-\sqrt{2}}\left(h(s) + \sum_{uv\in G}\|uv\|_2\right). \end{aligned}$$ ◀

Finally we will bound the total length of the greedy edges in terms of $\|st\|_\infty$. For any point u, let $\overline{u}$ be the projection of u onto d. Define the *footprint* of an edge uv in the following way. Let w be the point along either ray bounding the cone $C_i(u)$ that contains v such that $\|uw\|_2 = \|uv\|_2$. The footprint of uv is the segment $\overline{uw}$. See Figure 4. It does not matter which ray we chose since the projection of w would be the same in both cases. Note that we have $\|uv\|_2 = \sqrt{2}\|\overline{uw}\|_2$.

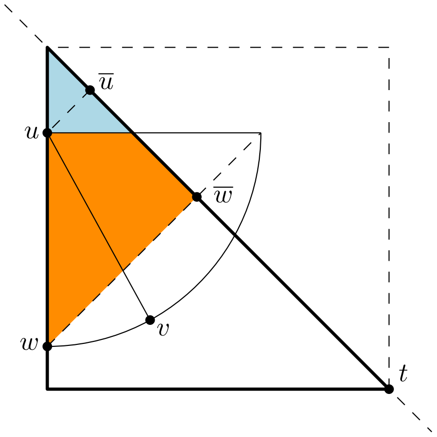


Figure 4 A greedy edge uv. Its footprint is the segment $\overline{uw}$. The blue triangle is empty because uv is a greedy edge, and the orange region is empty because it is contained in W_{uv}, the empty quarter circle defined by the edge uv. For any greedy edge $u'v'$ such that u' is in $C_1(t)$, u' is below d, and u' comes after u in the routing path, we must have u' in the indicated triangle. The footprint of $u'v'$ must be disjoint from the footprint of uv.

Recall that there are only two cones of t where greedy edges can originate, since in two of the cones of t a greedy step would be the same thing as a sweeping step. In both of these cones, the greedy edge can begin either above or below d. Split the set greedy edges into four subsets, depending on which cone of t the edge originates in and whether the edge begins above or below d. We will show that, for any two edges that are in the same subset of G, their footprints are disjoint (except possibly at a single point).

▶ **Lemma 7.** *Let uv and $u'v'$ be two greedy edges such that u and u' are in the same cone of t, and both are either above or below d. Then the footprints of these two edges are disjoint.*

Proof. Let $\overline{uw}$ and $\overline{u'w'}$ be the footprints of the edges uv and $u'v'$, respectively. Assume without loss of generality that u appears before u' on the routing path, and that u and u' are in $C_1(t)$ and below d. We will show that the points $\overline{u}$, $\overline{w}$, $\overline{u'}$, $\overline{w'}$, and t appear in exactly that order along d. To do this, we show that $\overline{uw}$ and $\overline{u'w'}$ are contained in $C_1(t)$ and that

$$\|\overline{u}t\|_1 > \|\overline{w}t\|_1 \geq \|\overline{u'}t\|_1 > \|\overline{w'}t\|_1. \tag{4}$$

First note that $\overline{w}$ must be in $C_1(t)$, since otherwise t would be in W_{uv}, which is empty. Recall that W_{uv} is the quarter-circle in $C_i(p)$ that is centred at p with q on its boundary contained in $C_i(p)$. Therefore we have $\|t\overline{u}\|_1 > \|t\overline{w}\|_1$. The same reasoning shows that $\|t\overline{u'}\|_1 > \|t\overline{w'}\|_1$.

Since u' comes after u on the routing path, we know that it is inside the square centred at t with u on its boundary by Lemma 2. The point u' must be inside the intersection of this square with $C_1(t)$, and below d. If $\|\overline{v}t\|_1 < \|\overline{u'}t\|_1$, then u' would either have to be inside $T(u, d)$ or W_{uv}. Both of these two regions are empty, so we must have $\|\overline{v}t\|_1 \geq \|\overline{u'}t\|_1$. See Figure 4. ◀

We can use this lemma and the fact that greedy edges can only originate in two cones of t to bound the total length of the greedy edges.

▶ **Lemma 8.** *The total L_2 length of the greedy edges is at most $8\|st\|_\infty$.*

Proof. Let $G_1, \ldots, G_4$ be the four sets of greedy edges defined above. The total length of the footprints in one such set cannot be more than $\sqrt{2}\|st\|_\infty$, since that is the length of d that lies inside the intersection of one of the cones of t with the square centred at t with s on its boundary. Therefore,

$$\sum_{uv \in G} \|uv\|_2 = \sum_{i=1}^{4} \sum_{uv \in G_i} \|uv\|_2 = \sum_{i=1}^{4} \sum_{uv \in G_i} \sqrt{2}\|\overline{uu}\|_2 \leq \sum_{i=1}^{4} 2\|st\|_\infty = 8\|st\|_\infty. \qquad ◀$$

Now that we have a bound on the length of the greedy edges in terms of $\|st\|_\infty$, we can combine that with our bound on the length of the sweeping edges to get a bound on the total length of the path, and therefore the routing ratio.

▶ **Theorem 9.** *The routing ratio of the greedy-sweep algorithm is at most* $17 + 9/\sqrt{2}$.

3.1 Removing the diagonal bit

In the greedy-sweep algorithm, one bit of memory is required to remember the choice of diagonal. Removing the single bit of memory just requires us to fix a diagonal ahead of time, without knowledge of s and t. We will choose $d_-(t)$ as our diagonal. The proof of Lemma 2 does not depend on our choice of d at all, so the routing algorithm will still terminate with this modification.

A few changes to the analysis have to be made, however. The initial height $h(s)$ can now be larger. The height of s for the one bit greedy-sweep is at most $\|st\|_\infty$, because we chose d to be whichever diagonal of t is closer to s. Now, however, since we fix the diagonal ahead of time the height of s is at most $2\|st\|_\infty$. The proofs of Lemmas 6 and 8 do not depend on the choice of diagonal, and so they remain unchanged. Notice how increasing $h(s)$ beyond $\|st\|_\infty$ also increases the distance $\|st\|_2$. This results in the routing ratio being a unimodal function of $\|st\|_\infty$, where at a certain point increasing $h(s)$ actually decreases the routing ratio since $\|st\|_\infty$ increases too quickly. The routing ratio will increase slightly, from $17 + 9/\sqrt{2} \approx 23.36$ to $\sqrt{331 + 154\sqrt{2}} \approx 23.43$.

▶ **Theorem 10.** *The memoryless version of the greedy-sweep algorithm has a routing ratio of at most* $\sqrt{331 + 154\sqrt{2}}$.

Proof. Assume without loss of generality that s lies on the left edge of the square centred at t with s on the boundary. By Lemmas 6 and 8 we know that the length of the routing path is at most $\alpha h(s) + (8\alpha + 8)\|st\|_\infty$, where $\alpha = 1 + \frac{1}{\sqrt{2}}$. If $h(s) \leq \|st\|_\infty$ then we can proceed as we did for the proof of Theorem 9.

If $h(s) > \|st\|_\infty$, then let θ be the angle made by the segment st with the ray $R_2(t)$ and let ρ be the routing ratio. Notice that $0 < \theta < \frac{\pi}{4}$. Then

$$\begin{aligned}\rho &\leq \frac{\alpha h(s) + (8\alpha + 8)\|st\|_\infty}{\|st\|_2} \\ &= \alpha \sin\theta + (9\alpha + 8)\cos\theta.\end{aligned}$$

This is maximized when $\theta = \arctan(\alpha/(9\alpha + 8))$ with a value of $\rho = \sqrt{331 + 154\sqrt{2}}$. ◀

4 Spanning ratio

We now turn our attention to the undirected Yao-4 graph. Let s and t be two vertices of a $\vec{Y}_4$ graph, and let $\mathcal{P}$ be the path from s to t constructed by the greedy-sweep algorithm. Notice that this is also a path in the undirected Y_4 graph. This already gives an upper bound of 23.36 on the spanning ratio of the undirected Y_4 graph. In this section, we will improve this upper bound to 16.95.

First, define $P_i(p)$ to be the path constructed by starting at p and following edges in cone i repeatedly, until a point is reached that has no neighbour in cone i. Also, given two points p and q define $R(p, q)$ to be the axis-aligned rectangle that has p and q at opposite corners. The side lengths of this rectangle are $d_x(p, q)$ and $d_y(p, q)$. Define $SS(p, q) = \min\{d_x(p, q), d_y(p, q)\}$ and $LS(p, q) = \max\{d_x(p, q), d_y(p, q)\}$.

Assume without loss of generality that s is in $C_1(t)$, and that it is below $d_-(t)$. In this situation, greedy edges can only originate in $C_1(t)$ and in $C_3(t)$, not $C_0(t)$ or $C_2(t)$. The key point of this section is that if any edge of $\mathcal{P}$ originates in $C_3(t)$, then $\mathcal{P}$ must intersect at least one of $P_0(t)$ and $P_2(t)$. We will show that if two edges of a Y_4 graph intersect, then there is a short path between the endpoints of the two edges. This means we can take a "shortcut" and only use the subpath of $\mathcal{P}$ before its intersection with $P_0(t)$ or $P_2(t)$.

Before proving the main result of this section, we will characterize the possible intersections in a Y_4 graph. First, some definitions. An edge pq is called an i-edge if q is in $C_i(p)$. If pq is an i-edge and uv is an $(i \pm 1)$-edge, then we say that pq and uv are in adjacent cones. If uv is an $(i + 2)$-edge, then we say that they are in opposite cones. Note that an i-edge is the same thing as an $(i \bmod 4)$-edge.

▶ **Lemma 11.** *Let pq and uv be i-edges such that p is not in $C_i(u)$ and u is not in $C_i(p)$. Then pq and uv cannot intersect except if $q = v$.*

The previous lemma implies that edges in the same cone can only intersect at their endpoints. We will characterize intersections of edges in opposite or adjacent cones into two categories: short side and long side crossings. See Figure 5.

If pq and uv are in opposite cones and pq intersects the short side of $R(p, u)$, then we say that *pq short side crosses uv*. Notice that two edges in opposite cones do not have to actually intersect for us to say that they short side cross. If the edges are in adjacent cones and pq intersects the short side of $R(p, v)$, then we say that pq short side crosses uv. If it intersects the long side of $R(p, v)$, then we say that *pq long side crosses uv*. Edges in adjacent cones must intersect if they short or long side cross. If the edges are in adjacent cones and $q = v$, then we will consider the intersection to be a short side crossing.

If pq and uv are in opposite cones and intersect, then we always consider this a short side crossing since one of the edges will short side cross $R(p, u)$, as the next lemma shows.

▶ **Lemma 12.** *Let pq and uv be edges in opposite cones that intersect. Then either pq short side crosses uv, or uv short side crosses pq.*

Note the slight difference in definitions for opposite and adjacent cones. If pq short side crosses uv and they are in adjacent cones, then pq intersects the short side of $R(p, v)$. But if they are in opposite cones, then pq intersects the short side of $R(p, u)$. We will call the second point (other than p) that defines this rectangle the *visible vertex*, so that we can say pq short side crosses an edge with visible vertex u.

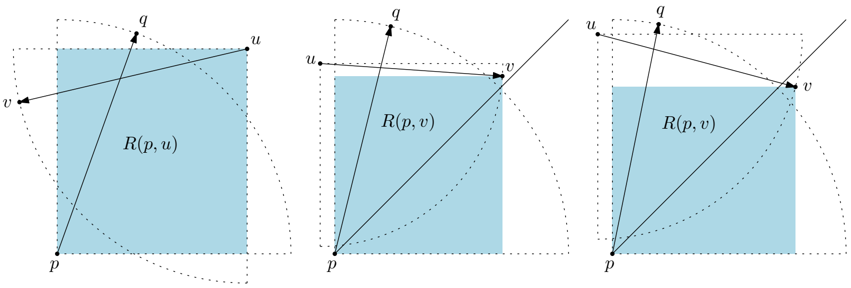


Figure 5 Three different crossings. From left to right: pq opposite cone short side crosses uv with visible vertex u, pq adjacent cone short side crosses uv with visible vertex v, and pq long side crosses uv.

4.1 Short side crossings

In this section we will prove that if an edge short side crosses another edge, then we can find a short undirected path between them. Long side crossings will be considered in the next section. The following two lemmas are simply geometric facts that will be needed to prove the main result of this section.

▶ **Lemma 13.** *Let p and q be two distinct points. Then $\|pq\|_2 \leq LS(p,q) + (\sqrt{2}-1)SS(p,q)$.*

Proof. Consider a right triangle with legs $LS(p,q)$ and $SS(p,q)$. Let θ be the angle adjacent to the hypotenuse and the leg with length $LS(p,q)$. We have $SS(p,q) = \|pq\|_2 \sin\theta$ and $LS(p,q) = \|pq\|_2 \cos\theta$.

$$(\sqrt{2}-1)SS(p,q) + LS(p,q) = \|pq\|_2((\sqrt{2}-1)\sin\theta + \cos\theta). \tag{5}$$

The function $(\sqrt{2}-1)\sin\theta + \cos\theta$ is at least 1 on the interval $[0, \pi/4]$. ◀

▶ **Lemma 14.** *Let pq be an edge that short side crosses an edge with visible vertex u. Then*
(1) *$LS(q,u) \leq SS(p,u)$, and*
(2) *$SS(q,u) \leq (\sqrt{2}-1)SS(p,u)$.*

Now we are ready to state the main result of this section.

▶ **Lemma 15.** *Let pq be an edge of a Y_4 graph that short side crosses another edge with visible vertex u. Then there is an undirected path from p to u with length at most $LS(p,u) + (\sqrt{2}+1)SS(p,u)$.*

Proof. The proof is by induction on all pairs of points $p \neq u$ such that there is an edge pq that short side crosses an edge with visible vertex u, ordered by $SS(p,u)$.

Consider one such pair. There are two possibilities. If $q = u$, then there is a path between p and u with length $\|pu\|_2 \leq \|pu\|_1 < LS(p,u) + (\sqrt{2}+1)SS(p,u)$. Otherwise $q \neq u$. Assume without loss of generality that pq is a 0-edge and that pq crosses the top edge of $R(p,u)$. Consider $P_3(q)$. We claim that this path must exit $R(q,u)$ by the right edge. Since u is in $C_3(q)$, the path must intersect some edge of $R(q,u)$.

If pq adjacent short side crosses an edge vu, then vu must be a 3-edge. Therefore $P_3(q)$ cannot intersect vu by Lemma 11, so it must intersect the right edge of $R(q,u)$ because vu is above the bottom edge. If pq opposite short side crosses an edge uv, then $R(p,u)$ must be empty since it is contained in the intersection of W_{pq} and W_{uv}. Let q' be the last point on $P_3(q)$. We have $d_y(p,q') < d_x(q',u)$, so the path $P_3(q)$ must exit $R(q,u)$ to the right.

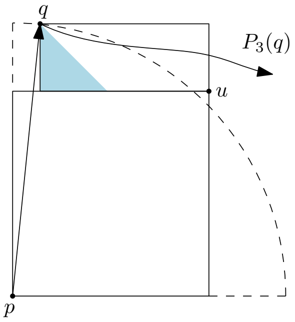


Figure 6 Figure for Lemma 15. Notice that $P_1(u)$ must either intersect pq or $P_3(q)$. If u' is the last vertex on $P_1(u)$ inside $R(q,u)$, then the long side of $R(q,u')$ is horizontal because the shaded region of $R(q,u')$ is contained in W_{pq}.

This implies that $P_1(u)$ must either intersect pq or $P_3(q)$. Both of these will result in a short side crossing. Let u' be the last point on $P_1(u)$ that is inside $R(q,u)$. If $P_1(u)$ intersects $P_3(q)$, there is a short side crossing since the edges will be in opposite cones, and edges in opposite cones that intersect always short side cross. Otherwise if $P_1(u)$ intersects pq, the long side of $R(q,u')$ is horizontal, so we have a short side crossing. Notice that this short side crossing has a smaller short side length than $SS(p,u)$. See Figure 6.

This implies that in the pair p and u such that $SS(p,u)$ is minimized, there is an edge from p to u. So in the base case there is a path between p and u with length at most $\|pu\|_2$.

Now assume for the inductive step that we have a pair of points p and u such that there is an edge pq that short side crosses an edge with visible vertex u, and that for every other such pair p' and u' such that $SS(p',u') < SS(p,u)$, there is a path from p' to u' of length at most $LS(p',u') + (\sqrt{2}+1)SS(p',u')$. If $P_1(u)$ intersects pq, then construct a path in the following manner. By induction, there is a path between q and u' with length at most $LS(q,u') + (\sqrt{2}+1)SS(q,u')$. Concatenate the edge pq, the path between q and u', and the segment of $P_1(u)$ up to u'. The resulting path has length at most

$$\begin{aligned} d_G(p,u) &\le \|pq\|_2 + LS(q,u') + (\sqrt{2}+1)SS(q,u') + \|uu'\|_1 \\ &\le \|pq\|_2 + LS(q,u) + (\sqrt{2}+1)SS(q,u). \end{aligned} \tag{6}$$

Now, if $P_1(u)$ intersects $P_3(q)$, then

$$\begin{aligned} d_G(p,u) &\le \|pq\|_2 + \|qq'\|_1 + LS(q',u') + (\sqrt{2}+1)SS(q',u') + \|uu'\|_1 \\ &\le \|pq\|_2 + LS(q,u) + (\sqrt{2}+1)SS(q,u). \end{aligned} \tag{7}$$

In both cases we have the same bound on the length of the path. Now, Lemma 13 implies that $\|pq\|_2 \le LS(p,u) + (\sqrt{2}-1)SS(p,u)$, and Lemma 14 gives bounds on $LS(q,u)$ and $SS(q,u)$. Putting these together finishes off the proof:

$$\begin{aligned} d_G(p,u) &\le \|pq\|_2 + LS(q,u) + (\sqrt{2}+1)SS(q,u) \\ &\le LS(p,u) + (\sqrt{2}-1)SS(p,u) + SS(p,u) + (\sqrt{2}+1)(\sqrt{2}-1)SS(p,u) \\ &= LS(p,u) + (\sqrt{2}+1)SS(p,u). \end{aligned}$$ ◀

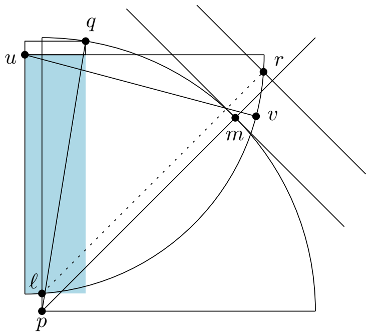


Figure 7 Figure for Lemma 17.

4.2 Long side crossings

Suppose we have an edge pq that long side crosses another edge uv. In this section, we will show that there is a short path from q to u, using the short side crossings from the previous section. Assume without loss of generality that uv is a 3-edge and pq is a 0-edge. We will show that from u and q we can find two paths in opposite cones that must intersect, and by Lemma 12 that intersection must be a short side crossing.

▶ **Lemma 16.** *$P_0(u)$ intersects the top edge of $R(u,q)$.*

Proof. The path $P_0(u)$ cannot intersect the right edge, since then it would have to intersect the edge pq. But pq is a 0-edge, and two 0-edges cannot intersect by Lemma 11. ◀

▶ **Lemma 17.** *$P_2(q)$ intersects the left edge of $R(u,q)$.*

Proof. Consider Figure 7. The point ℓ is directly above p and on the arc of W_{uv}. The point r is the other intersection of the line $d_+(\ell)$ with the arc of W_{uv}.

Every point on the arc of W_{pq} is below $d_-(m)$, including q. The point r is above $d_+(p)$ and on the arc of W_{uv}. This means that r must be above and to the right of m. Therefore q is also below $d_-(r)$.

Since q is above r but below $d_-(r)$, it must be the case that q is to the left of r. In other words, we have $d_x(u,q) < d_x(u,r) = d_x(u,y)$, which is what we wanted to show.

We have shown that $d_y(u,\ell) > d_x(u,q)$. That implies that the rectangle R with u as its upper-left corner, with width $d_x(u,q)$ and height $d_y(u,\ell)$, is taller than it is wide. Notice that R is contained in $W_{pq} \cup W_{uv}$, meaning that it is empty of points. Consider an edge xy on $P_2(q)$ such that x is in $R(u,q)$. If y is below ℓ, then we must have

$$\|xy\|_2 > d_y(x,y) > d_y(x,\ell) > d_y(x,u) + d_x(x,u) > \|xu\|_2,$$

a contradiction since both y and u are in $C_2(x)$. Therefore no edge of $P_2(q)$ can span R in this sense, and $P_2(q)$ must intersect the left edge of $R(u,q)$. ◀

These two lemmas imply the main result of this section.

▶ **Lemma 18.** *Let pq be an edge of a Y_4 graph that long side crosses another edge uv. Then there is an undirected path between u and q with length at most $LS(u,q) + (\sqrt{2}+1)SS(u,q)$.*

4.3 Constructing a path

In this section we describe how to construct a path between two vertices of an undirected Y_4 graph. Let s and t be the endpoints of our path. Let $\mathcal{P}$ be the path from s to t constructed by the greedy sweep algorithm in the directed $\vec{Y_4}$ graph. This is also a path in the undirected Y_4 graph. For the remainder of this section, assume without loss of generality that s is above and to the left of t, and that s is below the diagonal $d_-(t)$.

We consider three cases, based on the observation that if some vertex of $\mathcal{P}$ lies in $C_3(t)$, then $\mathcal{P}$ must intersect at least one of $P_0(t)$ or $P_2(t)$. We will construct a path in a different way for each of the three cases. The cases that we consider then are:

1. The path $\mathcal{P}$ does not intersect $P_0(t) \cup P_2(t)$, except for at t
2. The first intersection of $\mathcal{P}$ with $P_0(t) \cup P_2(t)$ is a short side crossing
3. The first intersection of $\mathcal{P}$ with $P_0(t) \cup P_2(t)$ is a long side crossing

In the first case, the path between s and t is $\mathcal{P}$ itself.

In the second case, let uv be the first edge of $\mathcal{P}$ that intersects $P_i(t)$, where $i \in \{0, 2\}$. If u is in $C_i(t)$, then uv must be an $(i+2)$-edge, so the edge pq that it intersects is in the opposite cone. Therefore by Lemma 15 there is a path between u and p with length at most $LS(p,u) + (\sqrt{2}+1)SS(p,u)$. Otherwise, if u is in $C_1(t)$, then uv must be a 3-edge and we have edges in adjacent cones that intersect. In this case we must have pq short side crossing uv, with visible vertex v. By Lemma 15 there is a path between v and p with length at most $LS(p,v) + (\sqrt{2}+1)SS(p,v)$. In either case concatenating the subpath of $\mathcal{P}$ from s to the visible vertex (u in the case of an opposite cone short side crossing, and v in the case of an adjacent cone short side crossing), the path from the visible vertex to p of Lemma 15, and the subpath of $P_i(t)$ from t to p (in reverse) gives a path between s and t.

The third case is similar to the second. Let uv be the first edge of $\mathcal{P}$ that intersects $P_i(t)$, where $i \in \{0, 2\}$. In this case pq long side crosses uv, meaning they must be in adjacent cones, and so uv is a 3-edge, and u is in $C_1(t)$. By Lemma 18 there is a path between u and q. Concatenate the subpath of $\mathcal{P}$ from s to u, the path from u to q, and the subpath of $P_i(t)$ from t to q (in reverse).

▶ **Theorem 19.** *Let P be a set of points. For any two points s and t in P, there is a path in $Y_4(P)$ with length at most $(13 + 5/\sqrt{2})\|st\|_2$.*

Proof. The path is constructed as previously described. We will bound the length for each case separately.

Case 1. Since no edge of P originates in $C_3(t)$, we know that any greedy edge of P will have to originate in $C_1(t)$. We can modify the proof of Lemma 8 using this fact. Now there are only two subsets of greedy edges to consider, depending on whether they originate above or below the diagonal. This results in a total length of $4\|st\|_\infty$ for the greedy edges. The proof of Lemma 6 does not need to change at all, giving a total length for P of

$$\begin{aligned}
\sum_{uv \in S} \|uv\|_2 + \sum_{uv \in G} \|uv\|_2 &\leq \left(1 + \frac{1}{\sqrt{2}}\right)\left(h(s) + \sum_{uv \in G} \|uv\|_2\right) + \sum_{uv \in G} \|uv\|_2 \\
&\leq \left(1 + \frac{1}{\sqrt{2}}\right)\left(\|st\|_\infty + 4\|st\|_\infty\right) + 4\|st\|_\infty \\
&\leq \left(9 + \frac{5}{\sqrt{2}}\right)\|st\|_2.
\end{aligned}$$

Case 2. Let u be the vertex of P that is the visible vertex of the first intersection of P with $P_i(t)$. Let P' be the subpath of P from s to u. Notice P' does not have any edges originating in $C_3(t)$. Therefore the total length of the greedy edges on this subpath is at most $4\|st\|_\infty$ as in case 1. Now we modify the proof of Lemma 6 by noting that

$$\sum_{pq\in D'}(h(p)-h(q)) = h(s)-h(u)+\sum_{pq\in I'}(h(q)-h(p)) \tag{8}$$

where D' and I' are the sets of edges of P' where the height decreases and increases, respectively, as in the proof of Lemma 6. Using that fact we see that the length of P' is at most $(8+4/\sqrt{2})\|st\|_\infty + (1+1/\sqrt{2})(h(s)-h(u))$.

Next, we know the length of the path between u and p by Lemma 15: at most $(\sqrt{2}+1)SS(u,p)+LS(u,p)$. And finally the length of the subpath of $P_i(t)$ is at most $\|pt\|_1$. Notice that p is inside $R(t,u)$, so we have

$$\begin{aligned}(\sqrt{2}+1)SS(u,p)+LS(u,p)+\|pt\|_1 &= (\sqrt{2}+1)SS(u,p)+LS(u,p)+SS(p,t)+LS(p,t)\\ &\le (\sqrt{2}+1)SS(t,u)+LS(t,u).\end{aligned}$$

Now, we have $SS(t,u)+LS(t,u)=h(u)$. Furthermore, $h(u)/2\le LS(t,u)\le h(u)$. Therefore,

$$\begin{aligned}(\sqrt{2}+1)SS(t,u)+LS(t,u) &= (\sqrt{2}+1)(h(u)-LS(t,u))+LS(t,u)\\ &= (\sqrt{2}+1)h(u)-\sqrt{2}LS(t,u)\\ &\le (\sqrt{2}+1)h(u)-\frac{\sqrt{2}}{2}h(u)\\ &= \left(1+\frac{1}{\sqrt{2}}\right)h(u).\end{aligned}$$

Adding this to the length of P', we see that the length of the path between s and t is at most

$$\begin{aligned}&\left(8+\frac{4}{\sqrt{2}}\right)\|st\|_\infty+\left(1+\frac{1}{\sqrt{2}}\right)(h(s)-h(u))+\left(1+\frac{1}{\sqrt{2}}\right)h(u)\\ &\quad= \left(8+\frac{4}{\sqrt{2}}\right)\|st\|_\infty+\left(1+\frac{1}{\sqrt{2}}\right)h(s)\\ &\quad\le \left(9+\frac{5}{\sqrt{2}}\right)\|st\|_\infty\\ &\quad\le \left(9+\frac{5}{\sqrt{2}}\right)\|st\|_2.\end{aligned}$$

Interestingly, this is the same bound as in case 1.

Case 3. Just like in case 2, the subpath P' from s to u has length at most $4\|st\|_\infty+(1+1/\sqrt{2})(4\|st\|_\infty+h(s)-h(u))\le(9+5/\sqrt{2})\|st\|_\infty$. By Lemma 18, the length of the path between u and q is at most $LS(u,q)+(\sqrt{2}+1)SS(u,q)$. Finally, the subpath of $P_i(t)$ from t to q has length at most $\|tq\|_1$.

We will show that $LS(u,q)\le d_y(u,p)$ and $SS(u,q)\le(\sqrt{2}-1)d_y(u,p)$. That will imply that the length of the subpath between u and q has length at most $d_y(u,p)+(\sqrt{2}+1)(\sqrt{2}-1)d_y(u,p)=2d_y(u,p)\le 2d_y(u,t)\le 2\|st\|_\infty$. We also have $\|qt\|_1\le 2\|qt\|_\infty\le 2\|st\|_\infty$. So the total length of the subpath between u and t is at most $4\|st\|_\infty$. Adding that to the length of P' gives a total of $(13+5/\sqrt{2})\|st\|_\infty$.

Consider Figure 8. Without loss of generality, we assume that pq is an edge on $P_0(t)$. Let $x=d_x(u,p)$ and $y=d_y(u,p)$. Let u' be the point directly above p such that $d_y(u',p)=y$.

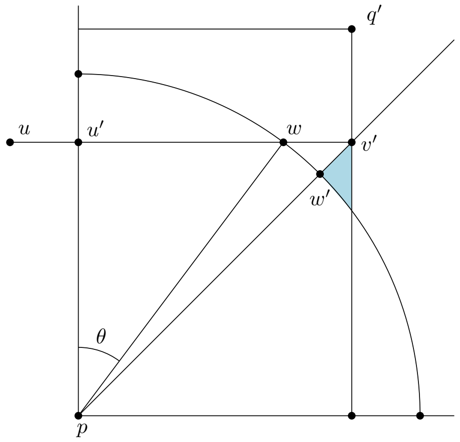


Figure 8 Case 3 of Theorem 19.

Let v' be the point of the bisector of $C_0(p)$ such that $d_x(v', p) = d_y(v', p) = y$. Let w be the point between u' and v' such that $d_x(w, v') = x$. Let w' be the point on the bisector of $C_0(t)$ such that $\|pw'\|_2 = \|pw\|_2$. Finally, let q' be the point above v' such that $d_y(p, q') = \|pv'\|_2$.

First, some observations. The point v must be to the left of v' since $\|uv'\|_2 = \|up\|_1 \geq \|uv\|_2$. The point q must be inside $R(u', q')$, since q is above u and right of p, but if it were outside of the rectangle then it would contain v' and the point v could not exist, because W_{pq} is empty of points.

We will show that $LS(u, q) \leq LS(u', q')$ and $SS(u, q) \leq SS(u', q')$. Then, since $LS(u', q') = y$ and $SS(u', q') = d_y(u', q') = (\sqrt{2} - 1)y$, we will have completed the proof.

First since q is inside $R(u', q')$ we must have $d_y(u, q) \leq d_y(u', q')$. Next we show that $d_x(u, q) \leq d_x(u', q')$. To do this we will show that q is to the left of w. Then we would have $d_x(u, q) \leq d_x(u, w) = d_x(u', q')$. We know that $\|uv\|_2 < \|up\|_2$, so if we can show that $\|up\|_2 < \|uw'\|_2$ then we will know that v cannot lie in the shaded region of Figure 8. That would mean that W_{pq} is inside W_{pw}, so $\|pq\|_2 < \|pw\|_2$, and since q is above w this must mean that q is left of w.

Now we prove that $\|up\|_2 < \|uw'\|_2$. Let $r = d_x(p, w') = d_y(p, w')$. Notice that $\|up\|_2^2 = x^2 + y^2$ and $\|uw'\|_2^2 = (x + r)^2 + (y - r)^2$. If we can show that the inequality $x^2 + y^2 < (x + r)^2 + (y - r)^2$ holds, we will be done. The inequality can be simplified to $x + r > y$. This holds since w is left of w', so $x + r = d_x(u, w') > d_x(u, w) = y$.

Therefore $\|uw'\|_2 > \|up\|_2$, which implies that q is to the left of w, which implies that $d_x(u, q) \leq d_x(u', q')$. This together with the fact that $d_y(u, q) \leq d_y(u', q')$ means we must have $LS(u, q) \leq LS(u', q') = d_y(u, p)$ and $SS(u, q) \leq SS(u', q') = (\sqrt{2} - 1)d_y(u, p)$. As we have previously shown, this means the length of the path between s and t has length at most $(13 + 5/\sqrt{2})\|st\|_2 \approx 16.54\|st\|_2$. Notice that the bound in this case is greater than the bound for the other two cases, by exactly $4\|st\|_2$. ◀

5 Conclusion

We have presented a local routing algorithm for the directed $\vec{Y}_4$ graph, the first such routing algorithm for this class of graphs. The routing ratio of this algorithm is $17 + 9/\sqrt{2} \approx 23.36$. The algorithm requires one bit of memory, and we showed that this can be dispensed with at the cost of increasing the routing ratio to $\sqrt{331 + 154\sqrt{2}} \approx 23.42$. Our result also lowers the best-known upper bound on the spanning ratio of the directed $\vec{Y}_4$ graph from 54.82 to

23.36. We also used the routing algorithm to bound the spanning ratio of the undirected Y_4 graph to be at most 16.54. To do so we showed that if two edges of a Y_4 graph intersect, then there must be a short path between the endpoints of these edges.

References

1 B. E. Flinchbaugh and L. K. Jones. Strong connectivity in directional nearest-neighbour graphs. *SIAM Journal on Algebraic Discrete Methods*, 2(4):461–463, 1981. doi:10.1137/0602049.

2 Luis Barba, Prosenjit Bose, Mirela Damian, Rolf Fagerberg, Wah Loon Keng, Joseph O'Rourke, André van Renssen, Perouz Taslakian, Sander Verdonschot, and Ge Xia. New and improved spanning ratios for Yao graphs. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 30–39, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2582112.2582143.

3 Prosenjit Bose, Jean-Lou De Carufel, Darryl Hill, and Michiel Smid. On the spanning and routing ratio of theta-four. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2361–2370. SIAM, 2019. doi:10.1137/1.9781611975482.144.

4 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 56–65, 1987. doi:10.1145/28395.28402.

5 Nawar M El Molla. *Yao spanners for wireless ad hoc networks.* Villanova University, 2009.

6 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. doi:10.1007/BF02189308.

7 William Michael Zoltan Kalnay. *Routing Ratio of the Directed Yao-6 Graphs.* Carleton University, 2023.

8 J. Mark Keil. Approximating the complete Euclidean graph. In *1st Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Nodes in Computer Science*, pages 208–213, 1988. doi:10.1007/3-540-19487-8_23.

9 Mirela Damian and Kristin Raudonis. Yao graphs span theta graphs. In *Combinatorial Optimization and Applications COCOA 2010*, volume 6509 of *Lecture Nodes in Computer Science*, pages 181–194, 2010. doi:10.1007/978-3-642-17461-2_15.

10 Mirela Damian and Naresh Nelavalli. Improved bounds on the stretch factor of Y_4. *Computational Geometry*, 62:14–24, 2017. doi:j.comgeo.2016.12.001.

11 Prosenjit Bose, Anil Maheshwari, Giri Narasimhan, Michiel Smid, and Norbert Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry*, 29(3):233–249, 2004. doi:10.1016/j.comgeo.2004.04.003.

12 Prosenjit Bose, Mirela Damian, Karim Douïeb, Joseph O'Rourke, Ben Seamone, Michiel Smid, and Stefanie Wuhrer. $\pi/2$-angle Yao graphs are spanners. In *Algorithms and Computation – 21st International Symposium, ISAAC 2010*, volume 6507 of *Lecture Nodes in Computer Science*, pages 446–457, 2010. doi:10.1007/978-3-642-17514-5_38.

13 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. doi:10.1137/0211059.

FPT Approximations for Fair k-Min-Sum-Radii

Lena Carta
University of Bonn, Germany

Lukas Drexler
Heinrich Heine University Düsseldorf, Germany

Annika Hennes[1]
Heinrich Heine University Düsseldorf, Germany

Clemens Rösner
Fraunhofer-Institut für Algorithmen und Wissenschaftliches Rechnen SCAI,
Sankt Augustin, Germany

Melanie Schmidt
Heinrich Heine University Düsseldorf, Germany

Abstract

We consider the k-min-sum-radii (k-MSR) clustering problem with fairness constraints. The k-min-sum-radii problem is a mixture of the classical k-center and k-median problems. We are given a set of points P in a metric space and a number k and aim to partition the points into k clusters, each of the clusters having one designated center. The objective to minimize is the sum of the radii of the k clusters (where in k-center we would only consider the maximum radius and in k-median we would consider the sum of the individual points' costs).

Various notions of fair clustering have been introduced lately, and we follow the definitions due to Chierichetti et al. [13] which demand that cluster compositions shall follow the proportions of the input point set with respect to some given sensitive attribute. For the easier case where the sensitive attribute only has two possible values and each is equally frequent in the input, the aim is to compute a clustering where all clusters have a 1:1 ratio with respect to this attribute. We call this the 1:1 case.

There has been a surge of FPT-approximation algorithms for the k-MSR problem lately, solving the problem both in the unconstrained case and in several constrained problem variants. We add to this research area by designing an FPT $(6 + \epsilon)$-approximation that works for k-MSR under the mentioned general fairness notion. For the special 1:1 case, we improve our algorithm to achieve a $(3 + \epsilon)$-approximation.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis

Keywords and phrases Clustering, k-min-sum-radii, fairness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.16

Related Version *Full Version*: https://arxiv.org/abs/2410.00598

Funding Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project 456558332

1 Introduction

Cluster analysis is an unsupervised learning task that has inspired much research during the last decades. Nearly all popular clustering formulations lead to NP-hard and often APX-hard problems, thus there is a thriving field designing approximation algorithms for clustering. A very popular and well studied problem is the k-*median* problem: Given n points P from a

[1] Corresponding author

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 16; pp. 16:1–16:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

metric space and a number k, find k centers $C \subseteq P$ such that the sum of the distances of all points to their respective centers is minimized. Notice that k centers implicitly define k clusters by assigning each point to its closest center (breaking ties arbitrarily). The k-median problem is APX-hard [19, 23], but allows for $O(1)$-approximations. After a long line of research, the currently best approximation for k-median achieves a guarantee of $2.675+\varepsilon$ [10][2]. A clustering function that is very popular for its simplicity and elegant algorithms is *k-center*. It has the same input and solution space, but judges clusterings based on the maximum radius of the (induced) clusters. The goal is to minimize the radius of the cluster with largest radius. It has been known for quite some time that k-center admits a 2-approximation [18, 20], and that this is tight when assuming P$\neq$ NP [21].

In this paper, we consider *k-min-sum-radii* clustering, abbreviated as k-MSR. We get the same input and solution space as for k-center, but the objective changes to the *sum* of the cluster radii. The problem thus lies in between k-center and k-median as it is a sum-based objective, but considers radii instead of points. Contrary to intuition, in metric spaces, many design techniques fail for k-msr which work fine for k-center and k-median. However, the problem allows for a polynomial $(3 + \epsilon)$-approximation [9] via primal dual algorithms.

The model of fair clustering was introduced by Chierichetti et. al. [13] based on a disparate impact approach. In the easiest case, given an input point set with the same number of blue and red points, the goal of fair clustering is to find a clustering where every cluster is composed of the same number of red and blue points (the colors represent values of a sensitive attribute). The number of points in a cluster is unlimited, but the composition of the clusters is constrained. For k-center *and* k-median, the following simple approach suffices to design polynomial-time approximation algorithms for fair clustering in this scenario:

Compute a fair micro clustering, i.e., a clustering into $\mu \gg k$ clusters which is fair (this is easier than finding exactly k clusters, for two colors and $\mu = n/2$ it is just a matching). Consider the clusters in this micro clustering as inseparable entities and use an algorithm for the unconstrained problem to cluster them into k clusters. The resulting clusters are fair because the union of fair clusters is again fair (this property of the constraint *fairness* is sometimes called *mergeability*). In addition, both for k-center and for k-median, the cost of the resulting clusters can be reasonably bounded, yielding $O(1)$-approximations for the respective fair clustering problems. Most surprisingly, the same approach does *not* work in the case of k-min-sum-radii. The reason is that for k-min-sum-radii, the cost of a micro clustering can actually be *larger* than the clustering with k clusters. This property is shared neither by k-center nor by k-median: For k-center, the radius of the micro clustering is always bounded by the k-clustering, and for k-median, the sum of the points' costs in the micro clustering is bounded by the respective sum of the k-clustering.

Figure 1 illustrates the situation for k-MSR. The figure is for unconstrained k-MSR to ease visualization, and just illustrates how the cost of micro clusterings for k-MSR can behave. The examples are depicted with $k = 1$ and $\mu = 4$. In (a), we see a cluster with k-MSR cost 1 in which all points have the same pairwise distance, namely 1. If this cluster is broken into μ pieces, then these pieces suddenly contribute t to the objective. In (b), we have a star with μ leaves where one cost 1 cluster remains when we use μ clusters, so the cost stays the same. In (c), the cluster only contains μ points, and then the cost drops to zero when it is divided into μ subclusters. Overall we observe that we have no control over the cost of a micro clustering and that the micro clustering approach fails.

[2] This result actually holds for a slightly more general variant where the set of input points P can be different than the set of possible center locations from which we pick C.

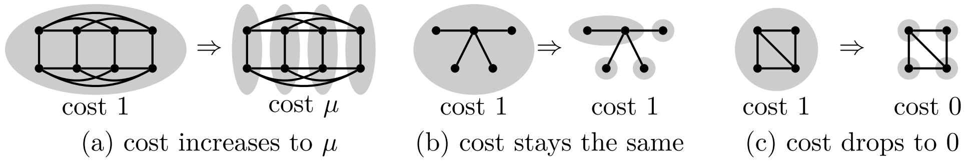


Figure 1 Anything can happen for k-MSR: The cost of a micro clustering with $\mu = 4$ compared to the macro clustering with $k = 1$. All pictures use shortest path metrics, the edges have unit weights.

Table 1 A list of FPT approximation results for k-MSR and k-median, all of which appeared in the last five years. Multiple results for the same problem are listed in reverse chronological order.

	Unconstr.	Capacities / Uniform Cap.	Matroid Con.	Fair Centers
k-MSR	$2+\epsilon$ [12]	≈ 7.6 [24], $15+\epsilon$ [5] / 3 [24], $4+\epsilon$ [5], 28 [22]	$9+\epsilon$ [22]	$3+\epsilon$ [12]
k-med	1.546 [4]$^{*)}$	$3+\epsilon$ [15], $7+\epsilon$ [1] / no improvement	$2+\epsilon$ [14]	–

$*$) This result holds for the problem as described in the introduction. If centers can be chosen from a set possibly different from P then the best known FPT approximation bound is $\approx 1.735+\epsilon$ [14].

FPT Approximation. There is mainly one approach for designing polynomial-time approximation algorithms for k-MSR, which is a primal-dual approach that yielded the currently best known bounds for unconstrained k-MSR and k-MSR with lower bounds on the cluster sizes, or outliers [9]. This lack of diversity in the techniques to obtain approximation algorithms for k-MSR has lately led to a surge of FPT approximation algorithms for k-MSR, that was also inspired by a similar strong interest in FPT approximation algorithms for the k-median problem. FPT approximation algorithms have been obtained for various problem variants, see Table 1. The "fair centers" variant demands that the set of centers is fair rather than the clusters. For more details, see the paragraph about related work. While the obtained approximation algorithms only work for small k, they are still of high interest due to the problem insights that they provide and also due to the fact that clustering with a small number of clusters is an important domain.

Our Results

We get the following results in FPT-time where the parameter is the number of clusters k.

- A $(3 + \epsilon)$-approximation for the fair clustering k-MSR problem when there are only two colors, both have exactly a ratio of 1:1 in the input point set, and we also want to achieve that exact ratio. We are not aware of any results on fair k-MSR in general metrics.
- A $(6 + \epsilon)$-approximation that works for a variety of more general fairness constraints, including all notions defined in [7, 13, 25]. To the best of our knowledge, there are no previous results for this problem.

We also extend our approach for clustering with uniform lower bounds, and generally to the class of *mergeable* constraints (see Definition 1). Uniform lower bounds have been studied in the clustering literature to model anonymity [2]: The constraint demands that every cluster contains a minimum number of ℓ points, i.e., that $|a(c)| \geq \ell$ for all $c \in C$. A polynomial-time $(3.5 + \epsilon)$-approximation algorithm for k-MSR with lower bounds is known [9], and our FPT approximation algorithm achieves a $(3 + \epsilon)$-guarantee.

Main Technical Contribution. Our main technical contribution is the development of a completely novel approach to design FPT approximation algorithms for k-MSR. We give more details on this approach in the next paragraph, but the main gain from our approach is that we make it possible to use k-center algorithms as a subroutine via a clever branching that we have not seen like this in the literature before. We believe this technique to be of independent interest for the design of FPT approximation algorithms for k-MSR.

Following this novel design scheme, it is possible to obtain $O(1)$-approximations for fair k-MSR and mergeable constraints in general. An additional technical contribution lies in reducing the factors to small constants. In particular in the general fairness case, obtaining the factor $6 + \epsilon$ requires a clever bounding technique.

More Insight into the Scheme. We first discuss another approach that does *not* work for k-MSR. There is a fairly general idea to obtain FPT approximations for constrained k-clustering problems which we can think of in the following way: Compute a solution to the unconstrained clustering problem with an approximation algorithm (here, one can even use a bicriteria approximation which computes $\mathcal{O}(k)$ centers). This gives a set of centers S. Then "move" all points to their closest center, i.e., create an instance I where all points lie at the k centers in S. The optimum cost for the constrained problem on I can be related to the optimum cost for the constrained problem on the original instance. Then solve I with an algorithm that uses that the points all lie on k locations (notice that the constraint will prevent us from simply opening one center at each location). The resulting solution is then translated back to the original instance.

Adamczyk et al. [1] use this approach to develop an FPT $(7 + \varepsilon)$-approximation for capacitated k-median. But here is another problem of the k-MSR cost function: Moving all points to centers of an approximate solution also has uncontrollable effects on the cost function. There seems to be no easy fix to this, and Inamdar and Varadarajan in the introduction of [22] also notice that the approach does not seem to extend to k-MSR, so there is a need for new techniques to design FPT algorithms for k-MSR in general.

The starting idea of our approach is to use an algorithm for the unconstrained *k-center* problem at the core of the algorithm (rather than for an unconstrained k-MSR problem, which would follow the above scheme). This means that we start off with a small approximation ratio of 2. Notice that there is a connection between the value F^* of an optimal k-center solution and the largest radius $r_{\max}$ in an optimal k-min-sum-radii solution: $r_{\max}$ must lie in the interval $[F^*, kF^*]$. This relation is obviously not tight enough to directly lead to an algorithm, but it can be used to find a near optimal approximation $\hat{r}_{\max}$ for the largest radius (a proof can be found in the full version). Now our first idea is that we can find the largest cluster in an k-MSR solution by running a k-center algorithm with $\hat{r}_{\max}$ and then guessing in which of the k-center clusters the largest k-MSR cluster lies. But this only works for the first cluster. To recurse, we have to eliminate this cluster from the input such that a k-center algorithm can find the next cluster. The main hurdle here is that we cannot simply delete the cluster because we might destroy the optimal fair assignment (which we do not know and cannot guess at this point). We resolve this problem by keeping all points but adjusting the metric to a (non-metric) distance function. Then we show that we can still solve the resulting problem by approximately solving a so called *k-center completion problem.*

This problem might be of independent interest and could play a role in further k-MSR research: Basically, we hand the problem an incomplete set of centers $C' = \{c_1, \ldots, c_\ell\}$ with $\ell \leq k$ together with radii $r_1, \ldots, r_\ell$ and ask it to find a k-center solution where points can be assigned to a center c_i from C' while paying r_i less than the actual distance. We observe that the k-center completion problem can be solved by Gonzalez' 2-approximation technique for k-center [18].

Using this modeling we are able to recursively find largest clusters. There is another technical problem, though. In every step, we are guessing the cluster in the completed solution returned from the k-center completion problem in which the optimal center of the next k-MSR cluster lies. But this may be one of the first ℓ clusters. Now the final crucial idea is that in this case, we opt to increase the radius of that previous cluster instead of opening a new cluster. It makes sense that for the k-MSR objective, this is a good idea: Often we can reduce cost by using less clusters instead of using overlapping clusters.

Combining these three ideas ((i) guessing clusters based on a k-center solution, (ii) modeling the elimination of clusters by using a completion problem, (iii) allowing clusters to grow), we obtain our main algorithm.

Applying our scheme yields a covering of the input points where we know that the balls are reasonably small compared to the balls in an optimal solution, and every optimum ball is covered by one of our balls. However, we also need to compute the actual clustering. Doing so requires to resolve the cluster membership of points that are in multiple balls, while respecting fairness constraints. For the general fairness case, we do this by modelling the overlap of balls by a graph and computing connected components in this graph. We can then show that the radius of the components is not too large (see Section 3.2).

Related Work. Research on FPT algorithms for constrained and unconstrained k-MSR is highly active at the moment, see Table 1. Notice that the paper by Chen et al. [12] gives an algorithm for a problem called *fair sum of radii*, but it refers to setting different from ours: While points also have colors, the fairness constraints are not imposed on the clusters but rather on the centers. More precisely, each color i has its own associated value k_i and any feasible solution has to use exactly k_i centers from that color. We call this *fair centers* in the table. All the results in the table are in general metric space, which is our setting.

In the more restricted Euclidean case, Drexler et al. [16] gave a PTAS for the fair k-MSR problem for constant k, however, the PTAS was faulty and a corrected version only exists for k-MSR with outliers, not for fair k-MSR. It is thus open to give a better result for Euclidean fair k-MSR. Bandyapadhyay et al. [5] give a $(2+\varepsilon)$-approximation for the capacitated k-MSR problem whose runtime linearly depends on the dimension, and a $(1+\varepsilon)$-approximation which depends exponentially on the dimension.

There are some results on poly-time approximation algorithms for k-MSR in general metrics, all following the primal-dual scheme. For the unconstrained case, the first was due to Charikar and Panigrahy [11], and there are two recent improvements by Friggstad and Jamshidian [17] and Buchem et al [9] (see below). The k-MSR problem with uniform lower bounds has been studied by Ahmadian and Swamy [3] who give a polynomial-time 3.83-approximation, and additionally give a 12.365-approximation if outliers are additionally considered. In [9], Buchem et al. improve upon all these factors by proposing a $(3+\varepsilon)$-approximation for the unconstrained case and the version with outliers, and a $(3.5+\varepsilon)$-approximation for lower bounds *and* lower bounds with outliers. Their algorithm also works for the non-uniform lower bounded case (for this case, we could achieve a $(6+\epsilon)$-approximation since lower bounds are mergeable, but that is worse than $(3.5+\epsilon)$).

2 Getting Started

Defining the k-Center and k-MSR Problem. An instance I for a k-clustering problem consists of a finite set P of n points, a (metric) distance function $d : P \times P \to \mathbb{R}_{\geq 0}$ and a number $k \in \mathbb{N}$ with $k \leq n$. A feasible solution $S = (\mathcal{C}, \sigma)$ consists of a set $\mathcal{C} = \{c_1, \ldots, c_k\} \subseteq P$

of *centers* and an assignment $\sigma : P \to \mathcal{C}$ of points to centers. For a center $c_i \in \mathcal{C}$, we call $C_i = \sigma^{-1}(c_i)$ the *cluster* of c_i induced by σ. Furthermore, we let $r_i = \max_{p \in C_i} d(p, c_i)$ denote the *radius* of C_i. Let $r_1, r_2, \ldots, r_k$ be the radii induced by a solution S. We refer to the tuple $(r_1, \ldots, r_k)$ as the *radius profile* of S. The cost of $S = (\mathcal{C}, \sigma)$ with radius profile $(r_1, \ldots, r_k)$ with respect to the k-center objective is defined as $\max_{i \in \{1,\ldots,k\}} r_i$, while the cost with respect to the k-min-sum-radii objective is defined as $\mathtt{MSR}(S) = \sum_{i=1}^{k} r_i$. The k-center/k-min-sum-radii problem takes as input a point set P, a metric $d \colon P \times P \to \mathbb{R}_{\geq 0}$ and a number $k \in \mathbb{N}$, and the task is to minimize the k-center/k-min-sum-radii objective.

Exact Fairness. In the fairness setting, every point belongs to exactly one protected group. Here, we will usually denote these groups by *colors* $\gamma_1, \ldots, \gamma_m$. A coloring of the points is given by a function $\gamma \colon P \to \{\gamma_1, \ldots, \gamma_m\}$.

The notion of *exact fairness* as for example defined in [25] is based on maintaining the underlying proportions of colors in the clusters. That is, for every color γ_j, the proportion of points in P with color γ_j is the same as the proportion of points with color γ_j in any cluster. To be more precise, we call a solution $S = (\mathcal{C}, \sigma)$ with induced clusters $C_1, \ldots, C_k$ *fair* if

$$\frac{|C_i \cap \Gamma_j|}{|C_i|} = \frac{|\Gamma_j|}{|P|}$$

for all $i \leq k$ and $j \leq m$, where $\Gamma_j = \gamma^{-1}(\gamma_j)$ is the set of points with color γ_j. The special case of $m = 2$ and $|\Gamma_1| = |\Gamma_2|$ has a specifically nice structure because the optimum solution can be partitioned into $|P|/2$ bicolored pairs. We refer to this as the 1:1 case.

There also exist more relaxed definitions of fairness that do not demand strict preservation of input ratios. In the full version, we discuss notions from [6, 7, 8, 13, 25].

Mergeable Constraints. Let $(\mathcal{C}, \sigma)$ be a clustering. We *merge* two clusters $C_i \coloneqq \sigma^{-1}(c_i)$, $C_j \coloneqq \sigma^{-1}(c_j)$ by replacing $c_i, c_j \in \mathcal{C}$ by an arbitrary point $c' \in C_i \cup C_j$ (i.e., $\mathcal{C} \coloneqq \mathcal{C} \setminus \{c_i, c_j\} \cup \{c'\}$) and assigning $\sigma(p) = c'$ for all $p \in C_i \cup C_j$. Notice that because σ maps every point to exactly one center, all clusters $C_i \coloneqq \sigma^{-1}(c_i),\ i = 1, \ldots, k$ need to be pairwise disjoint.

▶ **Definition 1.** *We say a constraint is* mergeable *if a feasible clustering is still feasible with respect to the constraint after merging two of its clusters.*

In this context, we say an assignment is *feasible* if it preserves the constraint. When dealing with the k-min-sum-radii problem with a specific mergeable constraint, we refer to the k-center problem with the same constraint as the *corresponding* k-center problem. For example, for k-min-sum-radii with exact fairness, the corresponding k-center problem is k-center with exact fairness. In the full version, we list several (fairness) constraints that are mergeable and prove this.

2.1 The k-center completion problem

The following problem can be solved in a relatively straightforward way using Gonzalez' algorithm [18].

▶ **Definition 2.** *The* k-center completion problem *takes as input a point set P, a metric $d \colon P \times P \to \mathbb{R}_{\geq 0}$, a number $k \in \mathbb{N}$, a set of predefined centers $c_1, \ldots, c_\ell$ for an $\ell \in [k]$, and corresponding radii $r_1, \ldots, r_\ell$. The aim is to compute centers $c_{\ell+1}, \ldots, c_k$ and an assignment $\alpha \colon P \to \{c_1, \ldots, c_k\}$ such that $\max_{x \in P} d'(x, \alpha(x))$ is minimized where*

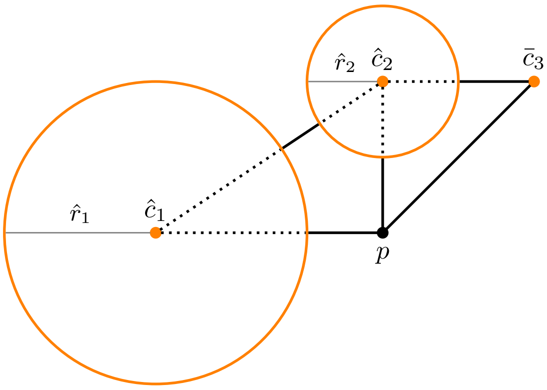


Figure 2 An instance of a 3-center completion problem. The centers $\hat{c}_1$ and $\hat{c}_2$ with corresponding radii $\hat{r}_1 = 1$ and $\hat{r}_2 = 0.5$ are already given. The underlying distances are given by $d(p, \hat{c}_1) = 1.5$, $d(p, \hat{c}_2) = 1$, $d(p, \bar{c}_3) = \sqrt{2}$. In d', all distances to one of the centers $\hat{c}_1, \hat{c}_2$ are shortened by the respective radius $\hat{r}_1, \hat{r}_2$. Dotted parts indicate the segments that do not contribute to the distance d'. For example, $d'(p, \hat{c}_1) = 0.5$, $d'(p, \hat{c}_2) = 0.5$. However, distances not involving $\hat{c}_1$ or $\hat{c}_2$ as one of the end points stay the same, i.e., $d'(p, \bar{c}_3) = d(p, \bar{c}_3)$. Originally, the point p is closer to $\bar{c}_3$ than to $\hat{c}_1$. But under d', p is closer to $\hat{c}_1$. This example also shows that the distance d' does not fulfill the triangle inequality: While $d'(p, \bar{c}_3) = \sqrt{2}$, the detour via $\hat{c}_2$ is shorter: $d'(p, \hat{c}_2) + d(\hat{c}_2, \bar{c}_3) = 1$.

$$d'(x,y) = \begin{cases} \max\{d(x,y) - r_i, 0\} & \textit{if } x \in P \setminus \{c_1, \dots, c_\ell\},\ y = c_i \textit{ with } i \in \{1, \dots, \ell\} \\ \max\{d(x,y) - r_i - r_j, 0\} & \textit{if } x = c_i,\ y = c_j \textit{ with } i, j \in \{1, \dots, \ell\} \\ d(x,y) & \textit{else} \end{cases}$$

i.e., the radius of clusters 1 *to* ℓ *is reduced by* $r_1, \dots, r_\ell$ *when computing the objective function.*

Figure 2 shows an example instance for such a completion problem. The k-center completion problem can be solved by running an adapted farthest-first traversal starting with $c_1, \dots, c_\ell$ and using distance function d'. For $i = \ell + 1, \dots, k$, always pick a point x that maximizes $\min_{j \in [i]} d'(x, c_j)$ and set $c_i := x$. When started with $\ell = 0$ and d instead of d', this is known as farthest-first traversal or Gonzalez' algorithm [18]. The change is that we already have chosen the first ℓ centers and thus they differ from what Gonzalez' algorithm would have picked (and consequently, the remaining centers also differ), and also that the distance to the first ℓ points is not metric. The adapted algorithm is described in Algorithm 1. Lemma 3 verifies that the algorithm still succeeds in computing a 2-approximation.

▶ **Lemma 3.** *Running Algorithm 1 with* d' *and* $c_1, \dots, c_\ell$ *already fixed yields a* 2*-approximation for the* k*-center completion problem with input* $(P, d, k, c_1, \dots, c_\ell, r_1, \dots, r_\ell)$*.*

Proof. We follow the proof for the approximation guarantee of Gonzalez' algorithm and verify that it still works in the case of the somewhat different k-center problem. Let D be the maximum distance of any point to its closest point in $\{c_1, \dots, c_k\}$ with respect to d', i.e., D is the cost of the solution computed by FARTHEST-FIRST-TRAVERSAL-COMPLETION with $c_1, \dots, c_\ell$ already fixed. Let c_{k+1} be a point with $\min_{i \in [k]} d'(c_{k+1}, c_i) = D$. Observe that all c_i with $i \geq \ell + 1$ satisfy that $d'(c_i, c_j) \geq D$ for all $j \in \{1, \dots, \ell\}$ because otherwise c_{k+1} would have been chosen as a center since its minimum distance is D. Inductively we also get for all c_i, c_j with $i, j \geq \ell + 1$ that $d'(c_i, c_j) \geq D$ is true because otherwise c_{k+1} would have been chosen. Now we get to the point where the proof differs slightly from the original

Algorithm 1 FARTHEST-FIRST-TRAVERSAL-COMPLETION.

```
Input   : Point set $P$, distance function $d$, integer $k$, centers $c_1, \dots, c_i$, radii $r_1, \dots, r_i$
Output  : Set of $k$ centers, assignment $\alpha$
// Update distance function
$d' \leftarrow d$
for $j = 1, \dots, i$ do
    for $p \in P$ do
        $d'(c_j, p) \leftarrow \max\{d(c_j, p) - r_j, 0\}$
// Complete centers by farthest-first-traversal
for $j = i + 1, \dots, k$ do
    $c_{i+1} \leftarrow \arg\max_{p \in P} \max_{c \in \{c_1, \dots, c_i\}} d'(p, c)$
// Assign points to their closest centers
for $p \in P$ do
    $\alpha(p) \leftarrow \arg\min_{c \in \{c_1, \dots, c_k\}} d'(p, c)$
return $c_1, \dots, c_k$, $\alpha$
```

proof because we have a case distinction. We have $k+1$ points $c_1, \dots, c_{k+1}$. In an optimum solution for the somewhat different k-center problem, we can assume that every point is assigned to its closest center, and in particular, all centers are assigned to themselves. There is always an optimum solution that satisfies this (this property is not necessarily ensured for clustering problems with constraints). Let $c^*_{\ell+1}, \dots, c^*_k$ and $\alpha^* \colon P \to \{c_1, \dots, c_\ell, c^*_{\ell+1}, \dots, c^*_k\}$ be a such an optimal solution, i.e., $\alpha^*(c_i) = c_i$ for all $i \in [\ell]$.

Case 1 is that for some $i \in \{\ell+1, \dots, k+1\}$, $\alpha^*(c_i) = c_j \in \{c_1, \dots, c_\ell\}$, i.e., one of the points we picked as a center or the additional point c_{k+1} is in the optimum solution assigned to one of the predefined centers. In this case, $OPT \geq d'(c_i, c_j) \geq D$ since we argued that all our centers have distance of at least D to the predefined centers.

Case 2 is that none of $c_{\ell+1}, \dots, c_{k+1}$ is assigned to any predefined center. Thus, they are all assigned to the $k - \ell$ centers $c^*_{\ell+1}, \dots, c^*_k$. By the pigeonhole principle, this means that $\alpha^*(c_i) = \alpha^*(c_j) = c^*_m$ for some $i, j \in \{\ell+1, \dots, k+1\}$ and $m \in \{\ell+1, \dots, k\}$. Since $i, j \in [\ell]$, $d'(c_i, c_j) \geq D$ as argued above. Also since $i, j, m \in [\ell]$, by definition, $d'(c_i, c_j) = d(c_i, c_j)$, $d'(c_i, c_m) = d(c_i, c_m)$ and $d'(c_j, c_m) = d(c_j, c_m)$.

We conclude by the triangle inequality that

$$\begin{aligned} D \leq d'(c_i, c_j) = d(c_i, c_j) &\leq d(c_i, c^*_m) + d(d^*_m, c_j) \\ &\leq 2 \max_{g=i,j} \{d(c_g, \alpha^*(c_g))\} \leq 2 \max_{g \geq \ell+1} \{d(c_g, \alpha^*(c_g))\}. \end{aligned}$$

As $d(c_g, \alpha^*(c_g)) = d'(c_g, \alpha^*(c_g))$ for all $g \geq \ell + 1$ by definition, this implies OPT $\geq \frac{1}{2}D$. ◀

2.2 Guessing an Approximate Radius Profile for the Optimum Solution

In the full version, we obtain the following corollary that allows us to guess close approximations for all radii of the optimal k-MSR solution in FPT time. Let $(r^*_1, \dots, r^*_k)$ be the radius profile of an optimal solution. We call a radius profile $(\tilde{r}_1, \dots, \tilde{r}_k)$ *near-optimal* if $r^*_i \leq \tilde{r}_i \leq (1+\varepsilon) r^*_i$ for all $i \in \{1, \dots, k\}$.

► **Corollary 4.** *Let $(r^*_1, \dots, r^*_k)$ be the radius profile of an optimal solution, and assume that we know the value of a constant-factor approximation solution for the corresponding k-center problem on the same instance. Then we can compute a set of size $O(\log^k_{1+\varepsilon}(k/\varepsilon))$ that contains a near-optimal radius profile $(\tilde{r}_1, \dots, \tilde{r}_k)$ in time $O(k \log^k_{1+\varepsilon}(k/\varepsilon))$.*

3 Algorithm for k-Min-Sum-Radii with Mergeable Constraints

The aim of this section is to prove the following Theorem 5 which is proven in Section 3.2, followed by Theorem 13 for 1:1 fairness in Section 3.2.1 and Corollary 14 for lower bounds in Section 3.2.2.

▶ **Theorem 5.** *For every $\varepsilon > 0$, there exists an algorithm that computes a $(6 - \frac{3}{k} + \varepsilon)$-approximation for k-min-sum-radii with mergeable constraints in time $O((k \log_{1+\varepsilon}(k/\varepsilon))^k \cdot \operatorname{poly}(n))$ if the corresponding constrained k-center problem has a constant factor approximation algorithm.*

Our algorithm works in two steps. First, the algorithm computes a candidate set of k radii and centers based on guessing. If it guesses correctly, the induced balls form a feasible k-min-sum-radii solution with certain properties. However, it might not fulfill the mergeable constraint yet. What it means to guess correctly is defined later in Definition 6 after the algorithm is specified. Notice that for this part, we need no assumptions about the constraint aside from the fact that an approximation algorithm for the k-center problem under this mergeable constraint exists. We need mergeability only in the computation of the final assignment in Section 3.2.

In the second step of the algorithm, we compute an assignment of points to the candidate centers. If the center and radius candidates from the first step are appropriate, then this assignment is guaranteed to fulfill the mergeable constraint.

In the following, we fix an optimal solution that we are trying to find. It consists of clusters $C_1^*, \dots, C_k^*$, with centers $c_1^*, \dots, c_k^*$ and radii $r_1^*, \dots, r_k^*$. We will assume that the optimal radii are sorted decreasingly. All clusters necessarily fulfill the mergeable constraint. Furthermore, we assume that we are in an iteration where we consider the radius profile $\tilde{r}_1, \dots, \tilde{r}_k$ satisfying $r_j^* \le \tilde{r}_j \le (1+\varepsilon) r_j^*$ for all $j \le k$. We also say that such a radius profile is near-optimal. Such an iteration exists due to Corollary 4. During this run of the algorithm, we are constructing candidate centers $\hat{c}_1, \dots, \hat{c}_k$ and candidate radii $\hat{r}_1, \dots, \hat{r}_k$. In summary, we have the following notation to be aware of during the following:

- $C_1^*, \dots, C_k^*$ denote an optimal clustering for k-MSR with mergeable constraint with centers $c_1^*, \dots, c_k^*$ and radii $r_1^* \ge \dots \ge r_k^*$
- $\tilde{r}_1, \dots, \tilde{r}_k$ denote initial near-optimal radius profile such that $r_j^* \le \tilde{r}_j \le (1+\varepsilon) r_j^*$ for all $j \le k$
- $\hat{c}_1, \dots, \hat{c}_i$, $\hat{r}_1, \dots, \hat{r}_i$ denote candidate centers and radii constructed up to iteration i

3.1 Selection of Candidate Centers and Radii

The general idea of the algorithm is as follows: Assume that in the beginning of iteration i, we already fixed candidate centers $\hat{c}_1, \dots, \hat{c}_{i-1}$ and candidate radii $\hat{r}_1, \dots, \hat{r}_{i-1}$. We compute a 2-approximation for the induced k-center completion instance. The resulting output consists of centers $\hat{c}_1, \dots, \hat{c}_{i-1}, \bar{c}_i, \dots, \bar{c}_k$, radii $\hat{r}_1, \dots, \hat{r}_{i-1}, \bar{r}_i, \dots, \bar{r}_k$ and an assignment α. We guess $\alpha(c_i^*)$, i.e. where the i-th center of the optimal solution is assigned to in the k-center completion solution. Recall that we already have a good approximation for $\tilde{r}_i$ for the corresponding optimal solution radius r_i^*. If we guess that $\alpha(c_i^*)$ is among the newly chosen centers, i.e. if $\alpha(c_i^*) = \bar{c}_j \in \{\bar{c}_i, \dots, \bar{c}_k\}$, we open a new ball with radius $\hat{r}_i = 3\tilde{r}_i$ at this center $\hat{c}_i := \bar{c}_j$. Otherwise, if we guess that $\alpha(c_i^*) = \hat{c}_j \in \{\hat{c}_1, \dots, \hat{c}_{i-1}\}$, there already exists a ball around this center, and we only need to enlarge this ball by $3\tilde{r}_i$. In order to have k centers in the end, we set $\hat{c}_i$ to some arbitrary point and $\hat{r}_i = 0$.

The guessing of the center assignments can be handled as follows. In every iteration i, we have k possible choices for $\alpha(c_i^*)$. So each sequence of k "guesses" can be encoded by a tuple $a = (a_1, \ldots, a_k) \in \{1, \ldots, k\}^k$, where $a_i = \ell$ means that in the ith iteration, we choose the ℓ-th center of the k-center completion solution as $\alpha(c_i^*)$ (i.e. $\hat{c}_\ell$ if $\ell < i$, or $\bar{c}_\ell$ if $\ell \geq i$). Thus, we can emulate the guessing by generating all such tuples upfront, computing the candidate balls for each of these, and choosing the best feasible one in the end. For a formal description of the algorithm, see Algorithm 2.

Algorithm 2 CENTERS-AND-RADII.

```
Input  : Points $P$, distances $d$, $k \in \mathbb{N}$, radius profile $(\tilde{r_1}, \ldots, \tilde{r_k})$, tuple $(a_1, \ldots, a_k)$
Output : Set of $k$ centers, set of $k$ radii
$I_0 \leftarrow (P, d, k, \emptyset, \emptyset)$
for $i = 1, \ldots, k$ do
    $(S_{kcc}, \alpha) \leftarrow$ FARTHEST-FIRST-TRAVERSAL-COMPLETION$(I_{i-1})$ , where
     $S_{kcc} = \{\hat{c}_1, \ldots, \hat{c}_{i-1}, \bar{c}_i, \ldots, \bar{c}_k\}$ and $\alpha \colon P \to S_{kcc}$
    if $a_i < i$ then
        // guess that $\alpha(c_i^*) \in \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$
        Set $\hat{r}_{a_i} \leftarrow \hat{r}_{a_i} + 3\tilde{r}_i$, choose $\hat{c}_i$ arbitrarily, set $\hat{r}_i \leftarrow 0$
    else if $a_i \geq i$ then
        // guess that $\alpha(c_i^*) \in \{\bar{c}_i, \ldots, \bar{c}_k\}$
        Set $\hat{c}_i \leftarrow \bar{c}_{a_i}, \hat{r}_i \leftarrow 3\tilde{r}_i$
    $I_i \leftarrow (P, d, k, \{\hat{c}_1, \ldots, \hat{c}_i\}, \{\hat{r}_1, \ldots, \hat{r}_i\})$
return $\{\hat{c}_1, \ldots, \hat{c}_k\}$, $\{\hat{r}_1, \ldots, \hat{r}_k\}$
```

In the full version, we show an example run of Algorithm 2. With this notation and Algorithm 2 in place, we can now formally define what it means to guess correctly.

▶ **Definition 6** (Guessing correctly). *Given a solution (S_{kcc}, α) for the k-center completion problem with input $\hat{c}_1, \ldots, \hat{c}_{i-1}$ and $\hat{r}_1, \ldots, \hat{r}_{i-1}$. We say that Algorithm 2 guesses correctly if the input tuple a is such that in every iteration i, a_i is a correct guess of the assignment of the next optimal center under α. To be more precise, a_i is the smallest index in $\{1, \ldots, k\}$ such that $c_{a_i} = \alpha(c_i^*)$ with $S_{kcc} = \{c_1, \ldots, c_k\}$, where c_i^* is the center of the next optimal cluster C_i^*.*

The idea of Algorithm 2 is that it fully covers one so-far uncovered optimal cluster in every iteration (under the assumption that the initial radius profile is near-optimal and Algorithm 2 guesses correctly). For the analysis, we need the following Lemma that bounds the cost of an optimal k-center completion solution in any iteration of Algorithm 2 by the radius of the largest remaining optimal cluster that is not fully covered yet. Combining with Lemma 3 gives an upper bound on the distance between an optimal center c^* and the center $\alpha(c^*)$ it is assigned to.

▶ **Lemma 7.** *Assume that up to the end of iteration i, Algorithm 2 chose centers $\hat{c}_1, \ldots, \hat{c}_i$ and radii $\hat{r}_1, \ldots, \hat{r}_i$ such that for all $p \in \bigcup_{j \leq i} C_j^*$, there exists a center $\hat{c}_\ell \in \{\hat{c}_1, \ldots, \hat{c}_i\}$ such that $d'(p, \hat{c}_\ell) = 0$. Then, the value of an optimal solution for the k-center completion problem with input $\{\hat{c}_1, \ldots, \hat{c}_i\}, \{\hat{r}_1, \ldots, \hat{r}_i\}$ is at most r_{i+1}^*.*

Proof. Consider the center extension $\{c_{i+1}^*, \ldots, c_k^*\}$. By the precondition of the lemma, we can assign every point in $p \in \bigcup_{j \leq i} C_j^*$ to a center in $\{\hat{c}_1, \ldots, \hat{c}_i\}$ at distance 0 with respect to d'. For all $h \geq i+1$, any $x \in C_h^*$ can be assigned to c_h^* at distance $\leq r_h^*$. As the optimal radii are sorted in decreasing order, r^*_{i+1} is the largest remaining radius among the optimal clusters under d'. Hence, the resulting assignment α' satisfies $d'(x, \alpha'(x)) \leq r^*_{i+1}$. Notice that $\{c^*_{i+1}, \ldots, c^*_k\}$ and α' form a feasible solution for the k-center completion problem and that the maximum radius of this solution is r^*_{i+1} as argued above. Hence, the optimum value for the k-center completion problem is upper bounded by r^*_{i+1}. ◀

Now, we can show that there exists a surjective mapping $\varphi\colon \{C^*_1, \ldots, C^*_k\} \to \hat{\mathcal{B}}$, where $\hat{\mathcal{B}}$ is the collection of balls from $\{B(\hat{c}_1, \hat{r}_1), \ldots, B(\hat{c}_k, \hat{r}_k)\}$ for which $\hat{r}_i > 0$, such that $C^*_j \subseteq \varphi(C^*_j)$ for all $j \leq k$. The next Lemma formalizes this.

▶ **Lemma 8.** *Assume that our guess of the initial radius profile is near-optimal and that Algorithm 2 guesses correctly. Let $\hat{\mathcal{B}}$ denote the set of balls $B(\hat{c}_1, \hat{r}_1), \ldots, B(\hat{c}_k, \hat{r}_k)$ found by Algorithm 2. Then the following two statements hold true*

1. *for all $j \leq k$, there exists $\ell \leq k$ such that $C^*_j \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$*
2. *for all $\ell \leq k$, if $r_\ell > 0$ then there exists $j \leq k$ such that $C^*_j \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$.*

Proof. We show that the statement holds at the end of every iteration of Algorithm 2. That is, we show that for all $i \leq k$, the following holds

(1) for all $j \leq i$, there exists $\ell \leq i$ such that $C^*_j \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$

(2) for all $\ell \leq i$, if $r_\ell > 0$ then there exists $j \leq i$ such that $C^*_j \subseteq B(\hat{c}_\ell, \hat{r}_\ell)$.

Then setting $i = k$ implies the result. We prove this via induction over $i \leq k$. For $i = 0$, the statements are trivially fulfilled. Now let the statement be fulfilled at the end of iteration $i - 1 < k$ for some $i > 0$.

By Lemma 7, $\mathrm{OPT}_{kcc} \leq r^*_i$, where OPT_{kcc} is the value of an optimal solution for the k-center completion problem that takes the centers and radii generated until the end of iteration $i - 1$ as input. By Lemma 3, Farthest-first-traversal-completion in Line 3 of Algorithm 2 computes a 2-approximation for the k-center completion problem. These two arguments together imply $d(c^*_i, \alpha(c^*_i)) \leq 2\,\mathrm{OPT}_{kcc} \leq 2r^*_i$.

If $c^*_i = \hat{c}_j$ for some $j \leq i - 1$, then for all $p \in C^*_i$, it is $d(p, \hat{c}_j) = d(p, c^*_i) \leq r^*_i \leq r^*_j$, where the last inequality holds because the optimal radii are sorted decreasingly.

If Algorithm 2 guesses correctly, $c_{a_i} = \alpha(c^*_i) = c^*_i = \hat{c}_j$ and the radius $\hat{r}^{\text{new}}_j$ produced in Line 6 fulfills $\hat{r}^{\text{new}}_j := \hat{r}_j + 3\tilde{r}_i \geq r^*_i$. Therefore C^*_i is covered completely by $B(\hat{c}_j, \hat{r}^{\text{new}}_j)$. This implies that (1) holds. For index i, the algorithm creates a new ball with radius $\hat{r}_i = 0$. Therefore, statement (2) is fulfilled by the induction hypothesis.

Now, we assume that $c^*_i \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$. There are two cases for the guess a_i.

1. Either $a_i < i$. Then, we are in Line 6 of Algorithm 2 and enlarge an already existing ball centered at $\alpha(c^*_i) = \hat{c}_{a_i}$ by $3\tilde{r}_i$, i.e., the a_i-th ball is $B(\hat{c}_{a_i}, \hat{r}_{a_i} + 3\tilde{r}_i)$ at the end of the iteration. For every $p \in C^*_i$,

   $$d(p, \hat{c}_{a_i}) \leq d(p, c^*_i) + d(c^*_i, \hat{c}_{a_i}) = d(p, c^*_i) + d'(c^*_i, \hat{c}_{a_i}) + \hat{r}_{a_i} = d(p, c^*_i) + d'(c^*_i, \alpha(c^*_i)) + \hat{r}_{a_i}.$$

   It is $d(p, c^*_i) \leq r^*_i$ as $p \in C^*_i$, and $d'(c^*_i, \alpha(c^*_i)) \leq 2r^*_i$ as α is the assignment given by the 2-approximation. Further, $r^*_i \leq \tilde{r}_i$. Overall, $d(p, \hat{c}_{a_i}) \leq 3\tilde{r}_i + \hat{r}_{a_i}$, which implies that the ball $B(\hat{c}_{a_i}, \hat{r}_{a_i} + 3\tilde{r}_i)$ covers C^*_i completely.
2. Or $a_i \geq i$. In this case, the algorithm creates a new ball $B(\hat{c}_i, \hat{r}_i)$ with $\hat{c}_i := c_{a_i} = \alpha(c^*_i)$ and $\hat{r} := 3\tilde{r}_i$. For every $p \in C^*_i$,

   $$d(p, \hat{c}_i) = d(p, \alpha(c^*_i)) \leq d(p, c^*_i) + d(c^*_i, \alpha(c^*_i)) = d(p, c^*_i) + d'(c^*_i, \alpha(c^*_i)),$$

   where the last equality holds because $c^*_i \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$ and a_i is the smallest index such that $c_{a_i} = \alpha(c^*_i)$, which implies $c_{a_i} \notin \{\hat{c}_1, \ldots, \hat{c}_{i-1}\}$. Again, $d(p, c^*_i) \leq r^*_i$ and $d'(c^*_i, \alpha(c^*_i)) \leq 2r^*_i$. Overall, $d(p, \hat{c}_i) \leq 3r^*_i \leq 3\tilde{r}_i$, and hence, C^*_i is completely covered by $B(\hat{c}_i, \hat{r}_i)$. ◀

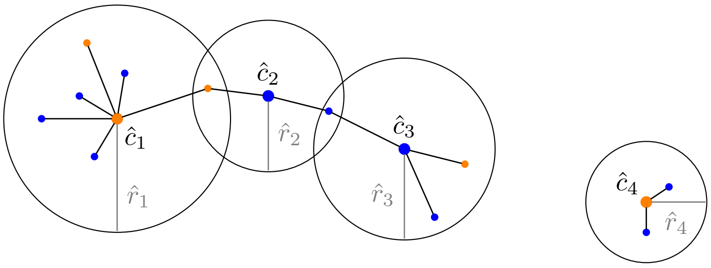


Figure 3 An instance of a k-min-sum-radii problem with exact fairness constraint with two colors and a blue:orange ratio of 2:1. The larger dots indicate centers and the gray lines indicate the radii output by Alg. 2. The black circles show the induced balls $B(\hat{c}_i, \hat{r}_i)$. The black lines between points represent the edges of the induced access graph. Note that the balls themselves are not necessarily fair, but every connected component is.

Notice that the candidate balls might overlap, but the optimal clusters are pairwise disjoint by definition of a clustering. The following lemma relates the total cost of the clustering consisting of the candidate balls to the cost of an optimal k-min-sum-radii with mergeable constraints solution. This will be useful for analyzing the cost of our final solution later on. Notice that this statement does not imply an approximation ratio for the vanilla k-min-sum-radii problem.

▶ **Lemma 9.** *Let $\hat{r}_1, \ldots, \hat{r}_k$ be the radii produced by Alg. 2. Then $\sum_{j=1}^k \hat{r}_j \leq 3(1+\varepsilon) \sum_{j=1}^k r_j^*$.*

Proof. We show by induction that $\sum_{j=1}^i \hat{r}_j \leq 3 \cdot \sum_{j=1}^i \tilde{r}_j$ for all $i \leq k$. Then for $i = k$, the result follows since $\tilde{r}_j \leq (1+\varepsilon) r_j^*$ for all $j \leq k$.

For $i = 0$, the statement trivially holds. Now assume that the statement holds for $i-1$. Either the algorithm sets $\hat{r}_i := 3\tilde{r}_i$ during iteration i. Then, $\sum_{j=1}^i \hat{r}_j = \sum_{j=1}^{i-1} \hat{r}_j + \hat{r}_i \leq 3 \sum_{j=1}^{i-1} \tilde{r}_j + 3\tilde{r}_i$. For the remaining case, let $\hat{r}_j^{(i-1)}$ denote the value of $\hat{r}_j$ at the beginning of the ith iteration, and $\hat{r}_j^{(i)}$ its value at the end of the iteration, $j \leq i$. There exists $\ell < i$ such that the algorithm sets $\hat{r}_\ell^{(i)} := \hat{r}_\ell^{(i-1)} + 3\tilde{r}_i$ and $\hat{r}_i^{(i)} := 0$. Then, $\sum_{j=1}^i \hat{r}_j^{(i)} = \sum_{j=1}^{i-1} \hat{r}_j^{(i)} = \sum_{j=1}^{i-1} \hat{r}_j^{(i-1)} + 3\tilde{r}_i \leq 3 \cdot \sum_{j=1}^i \tilde{r}_j$. ◀

3.2 Finding the Assignment

In the following, we will show how to find a feasible assignment. We construct a graph from center and radii candidates computed in the first part of the algorithm and observe that the clustering induced by the connected components of this graph fulfills the given mergeable constraint. We define the *access graph* $G = (V, E)$ as follows. The set of vertices corresponds to the given point set, i.e. $V = P$. We add an edge between any pair of vertices $x, y \in V$ iff $x = \hat{c}_i$ for a center $\hat{c}_i$ constructed in Algorithm 2 and $d(y, \hat{c}_i) \leq \hat{r}_i$ for the corresponding radius $\hat{r}_i$. The construction is exemplified in Figure 3. A connected component is a maximal connected subgraph of G. Let $\mathrm{CC}(G)$ denote the set of connected components in G. Covering a connected component $Z \in \mathrm{CC}(G)$ using one large cluster is not more expensive than covering it using the balls $B(\hat{c}, \hat{r})$ for all $\hat{c} \in Z$.

▶ **Lemma 10.** *Assume Algorithm 2 made the correct decision in each iteration and terminates with centers $\hat{C} = \{\hat{c}_1, \dots, \hat{c}_k\}$ and radii $\hat{r}_1, \dots, \hat{r}_k$. Let $G = (V, E)$ be the corresponding access graph. Let $Z \in \mathrm{CC}(G)$ be a connected component of G. Then assigning all vertices from Z to an arbitrary point in Z yields a cluster that is feasible with respect to the given mergeable constraint.*

Proof. Let $Z \in \mathrm{CC}(G)$ be a connected component of G. Let $\mathrm{V}(Z)$ denote the set of vertices of Z. We will show that $\mathrm{V}(Z)$ consists solely of ℓ optimal clusters that all lie entirely in $\mathrm{V}(Z)$ for some $\ell \geq 1$. As optimal clusters fulfill the mergeable constraint, the union of these also fulfills the mergeable constraint.

Every point $p \in \mathrm{V}(Z)$ lies in some optimal cluster. Hence, there exists at least one optimal cluster that intersects $\mathrm{V}(Z)$. We want to conclude that such a cluster already is completely contained in $\mathrm{V}(Z)$. Assume for a contradiction that there exists an optimal ball C^* such that $C^* \cap \mathrm{V}(Z) \neq \emptyset$ and $C^* \not\subseteq \mathrm{V}(Z)$. By Lemma 8, there exists a ball $B(\hat{c}, \hat{r})$ such that $C^* \subseteq B(\hat{c}, \hat{r})$. Let $v \in C^* \setminus \mathrm{V}(Z)$. Then $d(v, \hat{c}) \leq \hat{r}$ and therefore $\hat{c}$ must be part of the connected component Z, a contradiction. ◀

We can use this insight as follows: For every connected component $Z \in \mathrm{CC}(G)$, pick one of the centers $\hat{c} \in \hat{C} \cap \mathrm{V}(Z)$ that lie inside the connected component and assign all points in $\mathrm{V}(Z)$ to $\hat{c}$. This way, we get a solution that contains one cluster per connected component. To achieve the smallest possible cost guarantee, we set $\hat{c}$ with the largest corresponding $\hat{r}$ as the final center.

▶ **Lemma 11.** *Let $\hat{C} = \{\hat{c}_1, \dots, \hat{c}_k\}$, $\hat{r}_1, \dots, \hat{r}_k$ and $G = (V, E)$ as in Lemma 10. For every connected component Z, we choose the center $\hat{c}^Z \in \hat{C} \cap \mathrm{V}(Z)$ such that the corresponding radius $\hat{r}^Z$ is maximal among all radii in the connected component. Then, the solution $(\mathcal{C}, f)$ with $\mathcal{C} = \{\hat{c}^Z \mid Z \in \mathrm{CC}(G)\}$ and $f \colon P \to \mathcal{C}$ with $f(z) = \hat{c}^Z$ for all $z \in Z$ and for all connected components Z is a $(6 - \frac{3}{k} + \varepsilon)$-approximation for the k-min-sum-radii problem under a mergeable constraint.*

Proof. Since each cluster in the solution corresponds to exactly one connected component of G, Lemma 10 implies that the solution fulfills the mergeable constraint.

It remains to prove the approximation factor. Let $Z \in \mathrm{CC}(G)$ be a connected component in G. Let $v^Z \in \arg\max_{p \in Z} d(\hat{c}^Z, v^Z)$. There exists a path from $\hat{c}^Z$ to v^Z in Z. A shortest such path $\hat{c}^Z, v^1, \hat{c}_1^Z, v^2, \hat{c}_2^Z, \dots v^\ell, \hat{c}_{\ell_Z}^Z, v^Z$ with $\ell_Z \leq k$ alternatingly visits points in $\hat{C}$ and $\mathrm{V}(Z) \backslash \hat{C}$. Therefore, its length is bounded by $\sum_{i \leq \ell_Z} 2\hat{r}_i^Z - \hat{r}_{\max}^Z$, where $\hat{r}_{\max}^Z := \max_{i \colon \hat{c}_i \in Z} \hat{r}_i^Z$. The radius of a cluster with center $\hat{c}^Z$ is given by $d(v^Z, \hat{c}^Z)$. Hence, the sum of the radii of such clusters is bounded by

$$\sum_{Z \in \mathrm{CC}(G)} d(v^Z, \hat{c}^Z) \leq \sum_{Z \in \mathrm{CC}(G)} \left(\sum_{i \leq \ell_Z} 2\hat{r}_i^Z - \hat{r}_{\max}^Z \right) = \sum_{j=1}^{k} 2\hat{r}_j - \sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z$$

where the second equality holds because a graph's connected components are disjoint. There exists a connected component Z' such that $\hat{r}_{\max}^{Z'} = \max_{i \leq k} \hat{r}_i =: \hat{r}_{\max}$. Therefore, $\sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z \geq \hat{r}_{\max}$. Further, $\hat{r}_{\max} \geq \frac{1}{k} \sum_{j=1}^{k} \hat{r}_j$. Hence,

$$\sum_{j=1}^{k} 2\hat{r}_j - \sum_{Z \in \mathrm{CC}(G)} \hat{r}_{\max}^Z \leq (2 - \frac{1}{k}) \sum_{j=1}^{k} \hat{r}_j,$$

and by Lemma 9,

$$\left(2-\frac{1}{k}\right)\sum_{j=1}^{k}\hat{r}_j \le 3(1+\varepsilon)\left(2-\frac{1}{k}\right)\sum_{j=1}^{k} r_j^* = \left(6-\frac{3}{k}\right)(1+\varepsilon)\sum_{j=1}^{k} r_j^*. \qquad ◀$$

Algorithm 4 finds such a solution. Now we are ready to prove our main Theorem.

Algorithm 3 Assignment.

```
Input : Graph $G = (V, E)$, distance function $d$, set of centers $\hat{c}_1, \dots, \hat{c}_k$
Output: Set of $\le k$ centers $\mathcal{C}$, assignment $f$
$\mathcal{C} \leftarrow \emptyset$
for each connected component $Z$ of $G$ do
    find a center $\hat{c} \in Z \cap \hat{C}$ such that $\hat{r}$ is largest
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\hat{c}\}$
    for all $p \in Z$ do
        $f(p) \leftarrow \hat{c}$
return $\mathcal{C}$, $f$
```

Algorithm 4 k-min-sum-radii with mergeable constraints.

```
Input : Point set $P$, distance function $d$, $k \in \mathbb{N}$
Output: Set of $\le k$ centers $\mathcal{C}$, assignment $f$
$U \leftarrow (6 + \varepsilon)\max_{x,y \in P} d(x, y)$            // upper bound on the sum of radii cost
$\mathcal{R} \leftarrow$ set of radius profile guesses
forall $(\tilde{r_1}, \dots, \tilde{r_k}) \in \mathcal{R}$ do
    forall $a \in \{1, \dots, k\}^k$ do
        $(\{\hat{c}_1, \dots, \hat{c}_k\}, \{\hat{r}_1, \dots, \hat{r}_k\}) \leftarrow$ Centers-and-Radii$(P, d, k, (\tilde{r_1}, \dots, \tilde{r_k}), a)$
        compute the access graph $G$ based on $\{\hat{c}_1, \dots, \hat{c}_k\}$ and $\{\hat{r}_1, \dots, \hat{r}_k\}$
        $(\mathcal{C}, f) \leftarrow$ Assignment$(G, d, \{\hat{c}_1, \dots, \hat{c}_k\})$
        if $(\mathcal{C}, f)$ is feasible and MSR$(\mathcal{C}, f) < U$ then
            $(\mathcal{C}^*, f^*) \leftarrow (\mathcal{C}, f)$
            $U \leftarrow$ MSR$(\mathcal{C}, f)$
return $\mathcal{C}^*, f^*$
```

▶ **Theorem 5.** *For every $\varepsilon > 0$, there exists an algorithm that computes a $(6 - \frac{3}{k} + \varepsilon)$-approximation for k-min-sum-radii with mergeable constraints in time $O((k \log_{1+\varepsilon}(k/\varepsilon))^k \cdot \mathrm{poly}(n))$ if the corresponding constrained k-center problem has a constant factor approximation algorithm.*

Proof. We invoke Algorithm 2 for all possible guesses of radius profiles and center assignments. For each of these, we compute the access graph G and invoke Algorithm 3 to obtain a solution. By Lemma 11, this solution is feasible and a $(6 - \frac{3}{k} + \varepsilon)$-approximation, assuming that Algorithm 2 guesses correctly. Since it iterates over all possible guesses, we can be sure that in one of the iterations we do indeed guess correctly. In the end, we return the best solution found, whose cost can therefore be upper bounded by $(6 - \frac{3}{k} + \varepsilon)$ times the optimum.

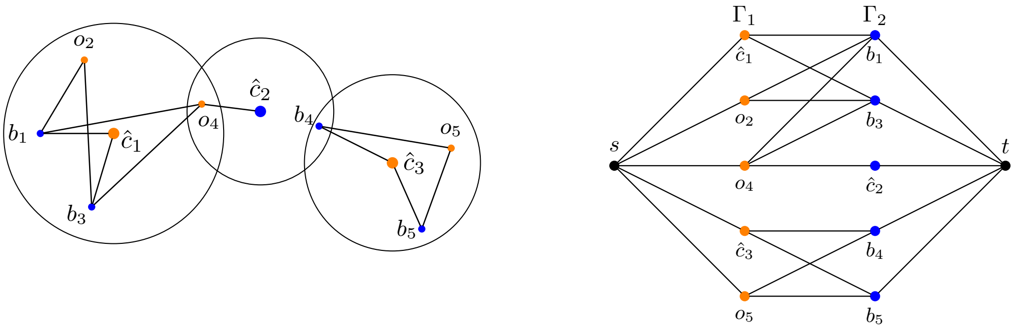


Figure 4 A fair k-min-sum-radii instance with two colors and equal proportions. Left: Output of Algorithm 2 as balls $B(\hat{c}_i, \hat{r}_i)$ for $i = 1, 2, 3$ and the graph edges between any fair pair of points that have access to the same center $\hat{c}$. Right: The corresponding flow network. All edges have capacity 1.

To be able to guess a radius profile, we first need to compute an approximate solution for constrained k-center, which can be done in polynomial time. By Corollary 4, we can then construct the set $\mathcal{R}$ in Line 2 of Algorithm 4 in time $O(k \log_{1+\varepsilon}(k/\varepsilon)^k)$. The outer for-loop in line 3 then goes through $|\mathcal{R}|$ iterations, which is $O(\log_{1+\varepsilon}(k/\varepsilon)^k)$. The inner for-loop in line 4 goes through k^k iterations. So in total, lines 5-10 are invoked $O\left((k \log_{1+\varepsilon}(k/\varepsilon))^k\right)$ times. The runtime of one call to Centers-and-radii is dominated by the runtime of the calls to Farthest-first-traversal-completion. This in turn has the same asymptotic running time as Gonzalez' algorithm, which can be implemented to run in $O(kn)$. So we can bound the runtime of Centers-and-radii by $O(k^2 n)$. The construction of the access graph can be performed in $O(kn)$, as can one call to Assignment. The feasibility of a solution can be checked in $O(n)$. Thus, we obtain an overall running time of $O\left((k \log(k/\varepsilon))^k \cdot \text{poly}(n)\right)$. ◀

3.2.1 Fairness with two Colors and Equal Proportions

We can get better guarantees for the exact fairness constraint with two colors and equal proportions. In this case, we can find a fair assignment such that none of the radii $\hat{r}$ has to be enlarged. The idea is that we first compute a fair micro clustering (i.e. partition P into fair pairs) and then assign these pairs to a common center.

Let $P = \Gamma_1 \cup \Gamma_2$, i.e., P consists of two different colors, γ_1, γ_2. We set $\Gamma_1 = \gamma^{-1}(\gamma_1)$, $\Gamma_2 = \gamma^{-1}(\gamma_2)$ as the sets of points carrying the respective color. Here, $|\Gamma_1| = |\Gamma_2| = \frac{n}{2}$. We want to partition the set P into pairs consisting of two points p_1, p_2 where $p_1 \in \Gamma_1$ and $p_2 \in \Gamma_2$. This is equivalent to finding a perfect matching of size $\frac{n}{2}$ between Γ_1 and Γ_2.

For this, we construct a flow network where there is an edge between $p_1 \in \Gamma_1$ and $p_2 \in \Gamma_2$ if and only if they have access to a common center $\hat{c}$, i.e., iff there exists $\hat{c} \in \hat{C}$ with $d(\hat{c}, p_1) \leq \hat{r}$ and $d(\hat{c}, p_2) \leq \hat{r}$. We connect all nodes of Γ_1 to some vertex s and all nodes of Γ_2 to some vertex t. We set the capacities of all the edges of this network to 1. Computing a perfect matching between points in Γ_1 and Γ_2 corresponds to finding a flow with value $\frac{n}{2}$ in the given network. Such a flow exists if Algorithm 2 guessed correctly.

From the flow, we can construct the fair pairs by combining two points $p_1 \in \Gamma_1$ and $p_2 \in \Gamma_2$ if the edge connecting them carries flow. We assign such a pair to a center $\hat{c}$ to which both points have access. We can summarize this in the following observation:

▶ **Observation 12.** *Let $F = \{p_1, p_2\}$ be a fair pair constructed as described above, let $\hat{c}$ be the center to which it gets assigned and $\hat{r}$ the corresponding radius. Then $\max_{i=1,2} d(p_i, \hat{c}) \leq \hat{r}$.*

In other words, assigning the fair pairs as a whole does not increase the cost. Together with Lemma 9, this implies the following theorem.

▶ **Theorem 13.** *Let $P = \Gamma_1 \cup \Gamma_2$ be a set of points consisting of only two different color groups Γ_1 and Γ_2 that fulfill $|\Gamma_1| = |\Gamma_2|$. For every $\varepsilon > 0$, there exists an FPT $(3+\varepsilon)$-approximation algorithm for the fair k-min-sum-radii problem.*

Proof. We run Algorithm 2 to obtain a set of balls $B(\hat{c}, \hat{r})$ and then compute a partitioning of P into fair pairs as described above. We assign each fair pair to a center to which both points of the pair have access. By Lemma 9, $\sum_{i=1}^{k} \hat{r}_i \leq 3(1+\varepsilon) \sum_{i=1}^{k} r_i^*$. As noted in Observation 12, assigning fair pairs does not increase the cost, which concludes the proof. ◀

3.2.2 Uniform Lower Bounds

In k-min-sum-radii with uniform lower bounds, we have an additional input number $\ell \in \mathbb{N}$, and every cluster in the solution needs to contain at least ℓ points. In this case we set up a network flow between the centers $\hat{C}$ on the left and the points on the right. There is a super source s and an edge $(s, \hat{c})$ for every $\hat{c} \in \hat{C}$. The capacity of these edges is set to the lower bound L. Then every $\hat{c}$ is connected to all $x \in P$ with $d(\hat{c}, x) \leq \hat{r}$. Finally, all points are connected with a unit capacity edge (x, t) to a super sink t. Any flow in this network corresponds to an assignment of at least L points to each center. Since our balls cover the optimum solution, we know that there exists a feasible flow in this network that sends $k \cdot L$ units of flow to the super sink. We can thus run a maximum flow algorithm to find such an assignment. After that, any remaining point $x \in P$ can be assigned arbitrarily to a center $\hat{c}$ with $d(x, \hat{c}) \leq \hat{r}$. Again, this is possible due to Lemma 8.

▶ **Theorem 14.** *There exists an FPT $(3+\varepsilon)$-approximation algorithm for the k-min-sum-radii problem with uniform lower bounds.*

References

1 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed Mohammad Meesum, and Michal Wlodarczyk. Constant-factor FPT approximation for capacitated k-median. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 1:1–1:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.1.

2 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Trans. Algorithms*, 6(3):49:1–49:19, 2010. doi:10.1145/1798596.1798602.

3 Sara Ahmadian and Chaitanya Swamy. Approximation algorithms for clustering problems with lower bounds and outliers. In *Proc. of the 43rd ICALP*, pages 69:1–69:15, 2016. doi:10.4230/LIPICS.ICALP.2016.69.

4 Aditya Anand and Euiwoong Lee. Separating k-median from the supplier version. In Jens Vygen and Jaroslaw Byrka, editors, *Integer Programming and Combinatorial Optimization - 25th International Conference, IPCO 2024, Wrocław, Poland, July 3-5, 2024, Proceedings*, volume 14679 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2024. doi:10.1007/978-3-031-59835-7_2.

5 Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT constant-approximations for capacitated clustering to minimize the sum of cluster radii. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.12.

6 Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4955–4966, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/fc192b0c0d270dbf41870a63a8c76c2f-Abstract.html.

7 Ioana Oriana Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 18:1–18:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.APPROX-RANDOM.2019.18.

8 Matteo Böhm, Adriano Fazzone, Stefano Leonardi, and Chris Schwiegelshohn. Fair clustering with multiple colors. *CoRR*, abs/2002.07892, 2020. arXiv:2002.07892.

9 Moritz Buchem, Katja Ettmayr, Hugo K. K. Rosado, and Andreas Wiese. A $(3+\varepsilon)$-approximation algorithm for the minimum sum of radii problem with outliers and extensions for generalized lower bounds. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1738–1765. SIAM, 2024. doi:10.1137/1.9781611977912.69.

10 Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 737–756. SIAM, 2015. doi:10.1137/1.9781611973730.50.

11 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 1–10. ACM, 2001. doi:10.1145/380752.380753.

12 Xianrun Chen, Dachuan Xu, Yicheng Xu, and Yong Zhang. Parameterized approximation algorithms for sum of radii clustering and variants. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 20666–20673. AAAI Press, 2024. doi:10.1609/AAAI.V38I18.30053.

13 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5029–5037, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

14 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.42.

15 Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.41.

16 Lukas Drexler, Annika Hennes, Abhiruk Lahiri, Melanie Schmidt, and Julian Wargalla. Approximating fair k-min-sum-radii in euclidean space. In Jaroslaw Byrka and Andreas Wiese, editors, *Approximation and Online Algorithms - 21st International Workshop, WAOA 2023, Amsterdam, The Netherlands, September 7-8, 2023, Proceedings*, volume 14297 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2023. doi:10.1007/978-3-031-49815-2_9.

17 Zachary Friggstad and Mahya Jamshidian. Improved polynomial-time approximations for clustering with minimum sum of radii or diameters. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 56:1–56:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.56.

18 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

19 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999. doi:10.1006/JAGM.1998.0993.

20 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2):180–184, 1985. doi:10.1287/MOOR.10.2.180.

21 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discret. Appl. Math.*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

22 Tanmay Inamdar and Kasturi R. Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *Proceedings of the 28th Annual European Symposium on Algorithms (ESA)*, volume 173, pages 62:1–62:17, 2020. doi:10.4230/LIPICS.ESA.2020.62.

23 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 731–740. ACM, 2002. doi:10.1145/509907.510012.

24 Ragesh Jaiswal, Amit Kumar, and Jatin Yadav. FPT approximation for capacitated sum of radii. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 65:1–65:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.65.

25 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 96:1–96:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.96.

Succinct Data Structures for Baxter Permutation and Related Families

Sankardeep Chakraborty
The University of Tokyo, Japan

Seungbum Jo[1]
Chungnam National University, Daejeon, Republic of Korea

Geunho Kim
Pohang University of Science and Technology, Republic of Korea

Kunihiko Sadakane
The University of Tokyo, Japan

Abstract

A permutation $\pi : [n] \to [n]$ is a Baxter permutation if and only if it does not contain either of the patterns 2–41–3 and 3–14–2. Baxter permutations are one of the most widely studied subclasses of general permutation due to their connections with various combinatorial objects such as plane bipolar orientations and mosaic floorplans, etc. In this paper, we introduce a novel succinct representation (i.e., using $o(n)$ additional bits from their information-theoretical lower bounds) for Baxter permutations of size n that supports $\pi(i)$ and $\pi^{-1}(j)$ queries for any $i \in [n]$ in $O(f_1(n))$ and $O(f_2(n))$ time, respectively. Here, $f_1(n)$ and $f_2(n)$ are arbitrary increasing functions that satisfy the conditions $\omega(\log n)$ and $\omega(\log^2 n)$, respectively. This stands out as the first succinct representation with sub-linear worst-case query times for Baxter permutations. The main idea is to traverse the Cartesian tree on the permutation using a simple yet elegant *two-stack algorithm* which traverses the nodes in ascending order of their corresponding labels and stores the necessary information throughout the algorithm.

Additionally, we consider a subclass of Baxter permutations called *separable permutations*, which do not contain either of the patterns 2–4–1–3 and 3–1–4–2. In this paper, we provide the first succinct representation of the separable permutation $\rho : [n] \to [n]$ of size n that supports both $\rho(i)$ and $\rho^{-1}(j)$ queries in $O(1)$ time. In particular, this result circumvents Golynski's [SODA 2009] lower bound result for trade-offs between redundancy and $\rho(i)$ and $\rho^{-1}(j)$ queries.

Moreover, as applications of these permutations with the queries, we also introduce the first succinct representations for mosaic/slicing floorplans, and plane bipolar orientations, which can further support specific navigational queries on them efficiently.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Succinct data structure, Baxter permutation, Mosaic floorplan, Plane bipolar orientation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.17

Related Version *Full Version*: `https://arxiv.org/abs/2409.16650` [8]

Funding *Seungbum Jo*: This work was supported by research fund of Chungnam National University.
Geunho Kim: This research was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)).

[1] Corresponding author

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 17; pp. 17:1–17:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A permutation $\pi : [n] \rightarrow [n]$ is a Baxter permutation if and only if there are no three indices $i < j < k$ that satisfy $\pi(j+1) < \pi(i) < \pi(k) < \pi(j)$ or $\pi(j) < \pi(k) < \pi(i) < \pi(j+1)$ (that is, π does not have pattern 2–41–3 or 3–14–2) [2]. For example, 3 5 2 1 4 is not a Baxter permutation because the pattern 2–41–3 appears ($\pi(2+1) = 2 < \pi(1) = 3 < \pi(5) = 4 < \pi(2) = 5$ holds). A Baxter permutation π is *alternating* if the elements in π rise and descend alternately. One can also consider *separable permutations*, which are defined as the permutations without two patterns 2–4–1–3 and 3–1–4–2 [7]. From the definitions, any separable permutation is also a Baxter permutation, but the converse does not hold. For example, 2 5 6 3 1 4 8 7 is a Baxter permutation but not a separable permutation because of the appearance of the pattern 2–4–1–3 (2 5 1 4).

In this paper, we focus on the design of a succinct data structure for a Baxter permutation π of size n, i.e., the data structure that uses up to $o(n)$ extra bits in addition to the information-theoretical lower bound along with supporting relevant queries efficiently. Mainly, we consider the following two fundamental queries on π: (1) $\pi(i)$ returns the i-th value of π, and (2) $\pi^{-1}(j)$ returns the index i of $\pi(i) = j$. We also consider the design of a succinct data structure for a separable permutation ρ of size n that supports $\rho(i)$ and $\rho^{-1}(j)$ queries. In the rest of this paper, log denotes the logarithm to the base 2, and we assume a word-RAM model with $\Theta(\log n)$-bit word size, where n is the size of the input. Also, we ignore all ceiling and floor operations that do not impact the final results.

1.1 Previous Results

For general permutations, there exist upper and lower bound results for succinct data structures supporting both $\pi(i)$ and $\pi^{-1}(j)$ queries in sub-linear time [20, 27]. However, to the best of our knowledge, there does not exist any data structures for efficiently supporting these queries on any subclass of general permutations. One can consider suffix arrays [21] as a subclass of general permutations, but their space consumption majorly depends on the entropy of input strings. This implies that for certain input strings, $\Omega(n \log n)$ bits (asymptotically the same space needed for storing general permutations) are necessary for storing the suffix arrays on them.

Baxter permutation is one of the most widely studied classes of permutations [5] because diverse combinatorial objects, for example, plane bipolar orientations, mosaic floorplans, twin pairs of binary trees, etc. have a bijection with Baxter permutations [1, 15]. Note that some of these objects are used in many applied areas. For example, mosaic floorplans are used in large-scale chip design [25], plane bipolar orientations are used to draw graphs in various flavors (visibility [34], straight-line drawing [17]), and floorplan partitioning is used to design a model for stochastic processes [29]. The number of distinct Baxter permutations of size n is $\Theta(8^n/n^4)$ [32], which implies that at least $3n - o(n)$ bits are necessary to store a Baxter permutation of size n. Furthermore, the number of distinct alternating Baxter permutations of size $2n$ (resp. $2n+1$) is $(c_n)^2$ (resp. $c_n c_{n+1}$) where $c_n = \frac{(2n)!}{(n+1)!n!}$ is the n-th Catalan number [11]. Therefore, at least $2n - o(n)$ bits are necessary to store an alternating Baxter permutation of size n. Dulucq and Guibert [12] established a bijection between Baxter permutations π of size n and a pair of unlabeled binary trees, called *twin binary trees*, which are essentially equivalent to the pair of unlabeled minimum and maximum Cartesian trees [35] for π. They provided methods for constructing π from the structure of twin binary trees and vice versa, both of which require $O(n)$ time. Furthermore, they presented a representation scheme that requires at most $8n$ bits for Baxter permutations of size n and $4n$

bits for alternating Baxter permutations of size n. Gawrychowski and Nicholson proposed a $3n$-bit representation that stores the tree structures of alternating representations of both minimum and maximum Cartesian trees [18]. Based on the bijection established in [12], the representation in [18] gives a succinct representation of a Baxter permutation of size n. Moreover, this representation can efficiently support a wide range of tree navigational queries on these trees in $O(1)$ time using only $o(n)$ additional bits. However, surprisingly, all of these previous representations of π crucially fail to address both, perhaps the most natural, $\pi(i)$ and $\pi^{-1}(j)$ queries efficiently as these queries have a worst-case time complexity of $\Theta(n)$.

Separable permutation was introduced by Bose et al.[7] as a specific case of patterns for the permutation matching problem. It is known that the number of separable permutations of size n equals the *large Schröder number* A_n, which is $\Theta\left(\frac{(3+2\sqrt{2})^n}{n^{1.5}}\right)$[36]. Consequently, to store a separable permutation ρ of size n, at least $n\log(3+2\sqrt{2})-O(\log n) \simeq 2.54n - O(\log n)$ bits are necessary. Bose et al. [7] also showed that ρ can be encoded as a *separable tree*, which is a labeled tree with at most $2n-1$ nodes. Thus, by storing the separable tree using $O(n\log n)$ bits, one can support both $\rho(i)$ and $\rho^{-1}(j)$ queries in $O(1)$ time using standard tree navigation queries. Yao et al. [36] showed a bijection between all canonical forms of separable trees with n leaves and the separable permutations of size n. To the best of our knowledge, there exists no $o(n\log n)$-bit representation for storing either separable permutations or their corresponding separable trees that can be constructed in polynomial time while supporting ρ queries in sub-linear time.

A mosaic floorplan is a collection of rectangular objects that partition a single rectangular region. Due to its broad range of applications, there is a long history of results (see [22, 36] and the references therein) concerning the representation of mosaic floorplans of size n in small space [1, 22, 23]. Ackerman et al. [1] presented a linear-time algorithm to construct a mosaic floorplan of size n from its corresponding Baxter permutation of size n and vice versa. Building on this construction algorithm, He [22] proposed the current state-of-the-art, a succinct representation of a mosaic floorplan of size n using $3n-3$ bits. Again, all of these previous representations primarily focus on constructing a complete mosaic floorplan structure and do not consider supporting navigational queries, e.g., return a rectangular object immediately adjacent to the query object in terms of being left, right, above, or below it, without constructing it completely. Note that these queries have strong applications like the placement of blocks on the chip [1, 37]. There also exists a subclass of mosaic floorplans known as *slicing floorplans*, which are mosaic floorplans whose rectangular objects are generated by recursively dividing a single rectangle region either horizontally or vertically. The simplicity of a slicing floorplan makes it an efficient solution for optimization problems, as stated in [38]. Yao et al. [36] showed there exists a bijection between separable permutations of size n and slicing floorplans with n rectangular objects. They also showed that separable trees can be used to represent the positions of rectangular objects in the corresponding slicing floorplans. However, to the best of our knowledge, there exists no representation of a slicing floorplan using $o(n\log n)$ bits that supports the above queries without reconstructing it.

1.2 Our Results and Main Idea

In this paper, we first introduce a $(3n+o(n))$-bit representation of a Baxter permutation π of size n that can support $\pi(i)$ and $\pi^{-1}(j)$ queries in $O(f_1(n))$ and $O(f_2(n))$ time respectively. Here, $f_1(n)$ and $f_2(n)$ are any increasing functions that satisfy $\omega(\log n)$ and $\omega(\log^2 n)$, respectively. We also show that the same representation provides a $(2n+o(n))$-bit representation of an alternating Baxter permutation of size n with the same query times. These are the first succinct representations of Baxter and alternating Baxter permutations that can support the queries in sub-linear time in the worst case.

Our main idea of the representation is as follows. To represent π, it suffices to store the minimum or maximum Cartesian tree defined on π along with their labels. Here the main challenging part is to decode the label of any node in either of the trees in sub-linear time, using $o(n)$-bit auxiliary structures. Note that all the previous representations either require linear time for the decoding or explicitly store the labels using $O(n \log n)$ bits. To address this issue, we first introduce an algorithm that labels the nodes in the minimum Cartesian tree in ascending order of their labels. This algorithm employs two stacks and only requires information on whether each node with label i is a left or right child of its parent, as well as whether it has left and/or right children. Note that unlike the algorithm of [12], our algorithm does not use the structure of the maximum Cartesian tree. We then proceed to construct a representation using at most $3n + o(n)$ bits, which stores the information used throughout our labeling algorithm. We show that this representation can decode the minimum Cartesian tree, including the labels on its nodes. This approach was not considered in previous succinct representations that focused on storing the tree structures of both minimum and maximum Cartesian trees, or their variants. To support the queries efficiently, we show that given any label of a node in the minimum or maximum Cartesian tree, our representation can decode the labels of its parent, left child, and right child in $O(1)$ time with $o(n)$-bit auxiliary structures. Consequently, we can decode any $O(\log n)$-size substring of the balanced parentheses of both minimum and maximum Cartesian trees with dummy nodes to locate nodes according to their inorder traversal (see Section 4.1 for a detailed definition of the inorder traversal) on π in $O(f_1(n))$ time. This decoding step plays a key role in our query algorithms, which can be achieved from non-trivial properties of our representation, and minimum and maximum Cartesian trees on Baxter permutations. As a result, our representation not only supports $\pi(i)$ and $\pi^{-1}(j)$ queries, but also supports range minimum/maximum and previous/next larger/smaller value queries efficiently.

Next, we give a succinct representation of separable permutation ρ of size n, which supports all the operations above in $O(1)$ time. Our result implies the Golynski's lower bound result [20] for trade-offs between redundancy and $\rho(i)$ and $\rho^{-1}(j)$ queries does not hold in separable permutations. The main idea of the representation is to store the separable tree of ρ using the *tree covering algorithm* [14], where each micro-tree is stored as its corresponding separable permutation to achieve succinct space. Note that a similar approach has been employed for succinct representations on some graph classes [4, 9]. However, due to the different structure of the separable tree compared to the Cartesian tree, the utilization of non-trivial auxiliary structures is crucial for achieving $O(1)$ query time on the representation.

Finally, as applications of our succinct representations of Baxter and separable permutations, we present succinct data structures of mosaic and slicing floorplans and plane bipolar orientations that support various navigational queries on them efficiently. While construction algorithms for these structures already exist from their corresponding Baxter or separable permutations [1, 6], we show that the navigational queries can be answered using a constant number of $\pi(i)$ (or $\rho(i)$), range minimum/maximum, and previous/next smaller/larger value queries on their respective permutations, which also require some nontrivial observations from the construction algorithms. This implies that our succinct representations allow for the first time succinct representations of these structures that support various navigation queries on them in sub-linear time. For example, we consider two queries on mosaic and slicing floorplans as (1) checking whether two rectangular objects are adjacent, and (2) reporting all rectangular objects adjacent to the given rectangular object. Note that the query of (2) was previously addressed in [1], as the *direct relation set* (DRS) query, which was computed in $O(n)$ time, and important for the actual placement of the blocks on the chip.

The paper is organized as follows. We introduce the representation of a Baxter permutation π of size n in Section 3. In Section 4, we explain how to support $\pi(i)$ and $\pi^{-1}(j)$ queries on π, in addition to tree navigational queries on both the minimum and maximum Cartesian trees. In Section 5.1, we present a succinct representation of separable permutation ρ that can support $\rho(i)$ and $\rho^{-1}(j)$ in $O(1)$ time. Finally, some preliminaries are outlined in the next section. Due to the space limit, the remaining results of our work (succinct representations of mosaic/slicing floorplans and plane bipolar orientations) are included in the full version of the paper [8].

2 Preliminaries

In this section, we introduce some preliminaries that will be used in the rest of the paper.

Cartesian trees. Given a sequence $S = (s_1, s_2, \ldots, s_n)$ of size n from a total order, a *minimum Cartesian tree* of S, denoted as $\mathsf{MinC}(S)$ is a binary tree constructed as follows [35]: (a) the root of the $\mathsf{MinC}(S)$ is labeled as the minimum element in S (b) if the label of the root is s_i, the left and right subtree of S are $\mathsf{MinC}(S_1)$ and $\mathsf{MinC}(S_2)$, respectively where $S_1 = (s_1, s_2, \ldots, s_{i-1})$ and $S_2 = (s_{i+1}, s_{i+2}, \ldots, s_n)$. One can also define a maximum Cartesian tree of S (denoted as $\mathsf{MaxC}(S)$) analogously. From the definition, in both $\mathsf{MinC}(S)$ and $\mathsf{MaxC}(S)$, any node with inorder i is labeled with s_i.

Balanced parentheses. Given an ordered tree T of n nodes, the BP of T (denoted as $BP(T)$) is defined as a sequence of open and closed parentheses constructed as follows [28]. One traverses T from the root node in depth-first search (DFS) order. During the traversal, for each node $p \in T$, we append “(” when we visit the node p for the first time, and append “)” when all the nodes on the subtree rooted at p are visited, and we leave the node p. From the construction, it is clear that the size of $BP(T)$ is $2n$ bits, and always balanced. Munro and Raman [28] showed that both (a) $\mathsf{findopen}(i)$: returns the position of matching open parenthesis of the close parenthesis at i, and (b) $\mathsf{findclose}(i)$: returns the position of matching close parenthesis of the open parenthesis at i, queries can be supported on $BP(T)$ in $O(t(n))$ time with $o(n)$-bit auxiliary structures, when any $O(\log n)$-bit substring of the $BP(T)$ can be decoded in $t(n)$ time. Furthermore, it is known that the wide range of tree navigational queries on T also can be answered in $O(t(n))$ time using $BP(T)$ with $o(n)$-bit auxiliary structures [30]: Here, each node is given and returned as the position of the open parenthesis that appended when the node is first visited during the construction of $BP(T)$ (for the full list of the queries, please refer to Table I in [30]).

Rank and Select queries. Given a sequence $S = (s_1, s_2, \ldots, s_n) \in \{0, \ldots, \sigma - 1\}^n$ of size n over an alphabet of size σ, (a) $\mathsf{rank}_S(a, i)$ returns the number of occurrence of $a \in \{0, \ldots, \sigma-1\}$ in $(s_1, s_2, \ldots, s_i)$, and (b) $\mathsf{select}_S(a, j)$ returns the first position of the j-th occurrence of $a \in \{0, \ldots, \sigma - 1\}$ in S (in the rest of this paper, we omit S if it is clear from the context). The following data structures are known, which can support both rank and select queries efficiently using succinct space [3, 31]: (1) suppose $\sigma = 2$, and S has m 1s. Then there exists a $(\log \binom{n}{m} + o(n))$-bit data structure that supports both rank and select queries in $O(1)$ time. The data structure can also decode any $O(\log n)$ consecutive bits of S in $O(1)$ time, (2) there exists an $(n \log \sigma + o(n))$-bit data structure that can support both rank and select queries in $O(1)$ time, and (3) if $\sigma = O(1)$ and one can access any $O(\log n)$-length sequence of S in $t(n)$ time, one can support both rank and select queries in $O(t(n))$ time using $o(n)$-bit auxiliary structures.

Range minimum and previous/next smaller value queries. Given a sequence $S = (s_1, s_2, \ldots, s_n)$ of size n from a total order with two positions i and j with $i \leq j$, the *range minimum query* $\mathsf{RMin}(i, j)$ on S returns the position of the smallest element within the range $s_i, \ldots, s_j$. Similarly, a *range maximum query* $\mathsf{RMax}(i, j)$ on S is defined to find the position of the largest element within the same range.

In addition, one can define *previous (resp. next) smaller value queries* at the position i on S, denoted as $\mathsf{PSV}(i)$ (resp. $\mathsf{NSV}(i)$), which returns the nearest position from i to the left (resp. right) whose value is smaller than s_i. If there is no such elements, the query returns 0 (resp. $n+1$). One can also define *previous (resp. next) larger value queries*, denoted as $\mathsf{PLV}(i)$ (resp. $\mathsf{NLV}(i)$) analogously.

It is known that if S is a permutation, RMin, PSV, and NSV queries on S can be answered in $O(1)$ time, given a BP of $\mathsf{MinC}(S)$ with $o(n)$ bit auxiliary structures [16, 30].

Tree Covering. Here, we briefly outline Farzan and Munro's [14] tree covering representation and its application in constructing a succinct tree data structure. The core idea involves decomposing the input tree into *mini-trees* and further breaking them down into smaller units called *micro-trees*. These micro-trees can be efficiently stored in a compact precomputed table. The shared roots among mini-trees enable the representation of the entire tree by focusing only on connections and links between these subtrees. We summarize the main result of Farzan and Munro's algorithm in the following theorem.

▶ **Theorem 1** ([14]). *For a rooted ordered tree with n nodes and a positive integer $1 \leq \ell_1 \leq n$, one can decompose the trees into subtrees satisfying the following conditions: (1) each subtree contains at most $2\ell_1$ nodes, (2) the number of subtrees is $O(n/\ell_1)$, (3) each subtree has at most one outgoing edge, apart from those from the root of the subtree.*

See Figure 3 for an example. After decomposing the subtree as above, any node with an outgoing edge to a child outside the subtree is termed a *boundary node*. The corresponding edge is referred to as the *non-root boundary edge*. Each subtree has at most one boundary node and a non-root boundary edge. Additionally, the subtree may have outgoing edges from its root node, designated as *root boundary edges*. For example, to achieve a tree covering representation for an arbitrary tree with n nodes, Theorem 1 is initially applied with $\ell_1 = \log^2 n$, yielding $O(n/\log^2 n)$ mini-trees. The resulting tree, formed by contracting each mini-tree into a vertex, is denoted as the *tree over mini-trees*. This tree, with $O(n/\log^2 n)$ nodes, can be represented in $O(n/\log n) = o(n)$ bits through a pointer-based representation. Subsequently, Theorem 1 is applied again to each mini-tree with $\ell_2 = \frac{1}{6}\log n$, resulting in a total of $O(n/\log n)$ micro-trees. The *mini-tree over micro-trees*, formed by contracting each micro-tree into a node and adding dummy nodes for micro-trees sharing a common root, has $O(\log n)$ vertices and is represented with $O(\log\log n)$-bit pointers. Encoding the non-root/root boundary edge involves specifying the originating vertex and its rank among all children. The succinct tree representation, such as balanced parentheses (BP) [26], is utilized to encode the position of the boundary edge within the micro-tree, requiring $O(\log \ell_2)$ bits. The overall space for all mini-trees over micro-trees is $O(n \log\log n/\log n) = o(n)$ bits. Finally, the micro-trees are stored with two-level pointers in a precomputed table containing representations of all possible micro-trees, demonstrating a total space of $2n + o(n)$ bits. By utilizing this representation, along with supplementary auxiliary structures that require only $o(n)$ bits of space, it is possible to perform fundamental tree navigation operations, such as accessing the parent, the i-th child, the lowest common ancestor, among many others, in $O(1)$ time [14].

3 Succinct Representation of Baxter Permutation

In this section, we present a $(3n + o(n))$-bit representation for a Baxter permutation $\pi = (\pi(1), \ldots, \pi(n))$ of size n. We begin by providing a brief overview of our representation. It is clear that the tree structure of $\mathsf{MinC}(\pi)$, along with the associated node labels can decode π completely. However, the straightforward storage of node labels uses $\Theta(n \log n)$ bits, posing an efficiency challenge. To address this issue, we first show that when π is a Baxter permutation, a two-stack based algorithm can be devised to traverse the nodes of $\mathsf{MinC}(\pi)$ according to the increasing order of their labels. After that, we present a $(3n + o(n))$-bit representation that stores the information used throughout the algorithm, and show that the representation can decode $\mathsf{MinC}(\pi)$ with the labels of the nodes.

Algorithm 1 Two-stack based algorithm.

```
Initialize two empty stacks $L$ and $R$.
Visit $\phi(1)$ (i.e., the root of $\mathsf{MinC}(\pi)$).
while $i = 2 \ldots n$ do
    // The last visited node is $\phi(i-1)$ by Lemma 2.
    if $\phi(i)$ is a left child of its parent then
        if $\phi(i-1)$ has a left child then
            Visit the left child of $\phi(i-1)$.
        else
            Pop a node from stack $L$, and visit the left child of the node.
        end
        if $\phi(i-1)$ has a right child that has not yet been visited then
            Push $\phi(i-1)$ to the stack $R$.
        end
    else // $\phi(i)$ is a right child of its parent
        if $\phi(i-1)$ has a right child then
            Visit the right child of $\phi(i-1)$.
        else
            Pop a node from stack $R$, and visit the right child of the node.
        end
        if $\phi(i-1)$ has a left child that has not yet been visited then
            Push $\phi(i-1)$ to the stack $L$.
        end
    end
end
```

Note that our encoding employs a distinct approach compared to prior representations, as seen in references [12, 13, 18, 24]. These earlier representations store the tree structures of $\mathsf{MinC}(\pi)$ and $\mathsf{MaxC}(\pi)$ (or their variants) together, based on the observation that there always exists a bijection between π and the pair of $\mathsf{MinC}(\pi)$ and $\mathsf{MaxC}(\pi)$ if π is a Baxter permutation [12]. We show that for any node in $\mathsf{MinC}(\pi)$, our representation allows to decode the labels of its parent, left child, and right child in $O(1)$ time using $o(n)$-bit auxiliary data structures. Using the previous representations that only store tree structures of $\mathsf{MinC}(\pi)$ and $\mathsf{MaxC}(\pi)$, these operations can take up to $\Theta(n)$ time in the worst-case scenario, even though tree navigation queries can be supported in constant time.

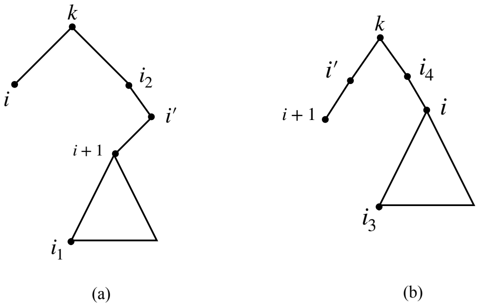


Figure 1 (a) the case when $\phi(i)$ is in the left subtree of $\phi(k)$, and (b) the case when $\phi(i)$ is in the right subtree of $\phi(k)$.

Now we introduce a two-stack based algorithm to traverse the nodes in $\mathsf{MinC}(\pi)$ according to the increasing order of their labels. Let $\phi(i)$ denote the node of $\mathsf{MinC}(\pi)$ with the label i. The algorithm assumes that we know whether $\phi(i)$ is left or right child of its parent for all $i \in \{2, \dots, n\}$.

The following lemma shows that if π is a Baxter permutation, the two-stack based algorithm works correctly.

▶ **Lemma 2.** *If π is a Baxter permutation, the two-stack based algorithm on $\mathsf{MinC}(\pi)$ traverses the nodes according to the increasing order of their labels.*

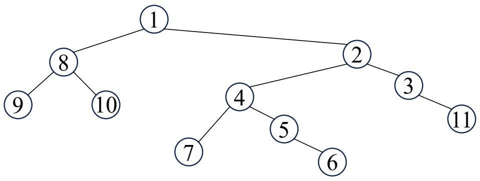
Proof. From Algorithm 1, it is clear that we first visit the root node, which is $\phi(1)$. Then we claim that for any i, the two-stack based algorithm traverses the node $\phi(i+1)$ immediately after traversing $\phi(i)$, thereby proving the theorem.

Suppose not. Then we can consider the cases as (a) the left child of $\phi(i)$ exists, but $\phi(i+1)$ is not a left child of $\phi(i)$, or (b) the left child of $\phi(i)$ does not exist, but $\phi(i+1)$ is not a left child of the node at the top of L. For the case (a) (the case (b) can be handled similarly), suppose $\phi(i+1)$ is a left child of the node $\phi(i')$. Then $i' < i$ by the definition of $\mathsf{MinC}(\pi)$ and the case (a). Now, let $\phi(k)$ be the lowest common ancestor of $\phi(i)$ and $\phi(i')$. If $\phi(i)$ is in the left subtree of $\phi(k)$ (see Figure 1(a) for an example), k cannot be i' from the definition of $\mathsf{MinC}(\pi)$. Then consider two nodes, $\phi(i_1)$ and $\phi(i_2)$, which are the leftmost node of the subtree rooted at node $\phi(i')$ and the node whose inorder is immediately before $\phi(i_1)$, respectively. Since $\phi(i_2)$ lies on the path from $\phi(k)$ to $\phi(i')$, we have $i+1 \leq i_1$ and $k \leq i_2 < i'$. Therefore, there exists a pattern 3–14–2 induced by $i - i_2, i_1 - i'$, which contradicts the fact that π is a Baxter permutation.

If $\phi(i)$ is in the right subtree of $\phi(k)$ (see 1(b) for an example), k cannot be i from the definition of $\mathsf{MinC}(\pi)$. Consider two nodes, $\phi(i_3)$ and $\phi(i_4)$, which are the leftmost node of the subtree rooted at node $\phi(i)$ and the node whose inorder is immediately before $\phi(i_3)$, respectively. Since $\phi(i_4)$ lies on the path from $\phi(k)$ to $\phi(i)$, we have $i+1 < i_3$ (i_3 is greater than i and cannot be $i+1$) and $k \leq i_4 < i$. Therefore, there exists a pattern 3–14–2 induced by $(i+1) - i_4, i_3 - i$, which contradicts the fact that π is a Baxter permutation.

The case when $\phi(i+1)$ is a right child of its parent can be proven using the same argument by showing that if the algorithm fails to navigate $\phi(i+1)$ correctly, the pattern 2–41–3 exists in π. ◀

The representation of π encodes the two-stack based algorithm as follows. First, to indicate whether each non-root node is whether a left or right child of its parent, we store a binary string $\mathsf{lr}[1, \dots n-1] \in \{l, r\}^{n-1}$ of size $n-1$ where $\mathsf{lr}[i] = l$ (resp. $\mathsf{lr}[i] = r$) if the node $\phi(i+1)$ is a left (resp. right) child of its parent. Next, to decode the information on



	1	2	3	4	5	6	7	8	9	10	11
lr	*r*	*r*	*l*	*r*	*r*	*l*	*l*	*l*	*r*	*r*	
lp	(	(	)	(		)	)				
rp			{					{	}	}	
E	3	3	2	3	2	0	0	3	0	0	
lrp	(	(	{)	(	[]	)	)	{	}	}	
U	1	1	2	1	2	1	1	1	1	1	

	1	2	3	4	5	6	7	8	9	10	11
permutation	9	8	10	1	7	4	5	6	2	3	11

Figure 2 An example of the representation of the Baxter permutation $\pi = (9, 8, 10, 1, 7, 4, 5, 6, 2, 3, 11)$. Note that the data structure maintains only E and lr along with $o(n)$-bit auxiliary structures.

the stack L during the algorithm, we define an imaginary string of balanced parentheses $\mathsf{lp}[1 \dots n-1]$ as follows: After the algorithm traverses $\phi(i)$, $\mathsf{lp}[i]$ is (1) "(" if the algorithm pushes $\phi(i)$ to the stack L, (2) ")" if the algorithm pops a node from the stack L, and (3) undefined otherwise. We also define an imaginary string of balanced parentheses $\mathsf{rp}[1, \dots n]$ in the same way to decode the information on the stack R during the algorithm. We use "{" and "}" to denote the parentheses in rp. Then from the correctness of the two-stack algorithm (Lemma 2), and the definitions of lr, lp, and rp, we can directly derive the following lemma:

▶ **Lemma 3.** *For any $i \in \{1, \dots, n-1\}$, the following holds:*

- *Suppose the node $\phi(i)$ is a leaf node. Then either $\mathsf{lp}[i]$ or $\mathsf{rp}[i]$ is defined. Also, $\mathsf{lr}[i]$ is l (resp. r) if and only if $\mathsf{lp}[i]$ (resp. $\mathsf{rp}[i]$) is a closed parenthesis.*
- *Suppose the node $\phi(i)$ only has a left child. In this case, $\mathsf{lr}[i]$ is l if and only if both $\mathsf{lp}[i]$ and $\mathsf{rp}[i]$ are undefined. Also, $\mathsf{lr}[i]$ is r if and only if $\mathsf{lp}[i] =$ "(" and $\mathsf{rp}[i] =$ "}".*
- *Suppose the node $\phi(i)$ only has a right child. In this case, $\mathsf{lr}[i]$ is l if and only if $\mathsf{lp}[i] =$ ")" and $\mathsf{rp}[i] =$ "{". Also, $\mathsf{lr}[i]$ is r if and only if both $\mathsf{lp}[i]$ and $\mathsf{rp}[i]$ are undefined.*
- *Suppose the node $\phi(i)$ has both left and right child. In this case, $\mathsf{lr}[i]$ is l if and only if $\mathsf{lp}[i]$ is undefined and $\mathsf{rp}[i] =$ "{". Also, $\mathsf{lr}[i]$ is r if and only if $\mathsf{lp}[i] =$ "(" and $\mathsf{rp}[i]$ is undefined.*

To indicate whether each node $\phi(i)$ has a left and/or right child we store a string $E \in \{0, 1, 2, 3\}^{n-1}$ of size $n-1$ where (a) $E[i] = 0$ if $\phi(i)$ is a leaf node (b) $E[i] = 1$ if $\phi(i)$ has only a left child, (c) $E[i] = 2$ if $\phi(i)$ has only a right child, and (d) $E[i] = 3$ if $\phi(i)$ has both left and right children. We store E using $2n + o(n)$ bits, which allows support both rank and select operations in $O(1)$ time [3]. Thus, the overall space required for our representation is at most $3n + o(n)$ bits ($2n$ bits for E, n bits for lr along with $o(n)$-bit auxiliary structures). From Lemma 3, our representation can access $\mathsf{lp}[i]$ and $\mathsf{rp}[i]$ in $O(1)$ time by referring $\mathsf{lr}[i]$ and $E[i]$. Next, we show both $\mathsf{findopen}$ and $\mathsf{findclose}$ on lp and rp in $O(1)$ time using the representation. We define an imaginary string lrp of length at most $2(n-1)$ over an alphabet of size 6 that consists of three different types of parentheses (), {}, [] constructed as follows. We first initialize lrp as an empty string and scan lr and E from the leftmost position. Then based on Lemma 3, whenever we scan $\mathsf{lr}[i]$ and $E[i]$, we append the parentheses to lrp as follows:

$$\begin{cases} (& \text{if } \mathsf{lr}[i] = r \text{ and } E[i] = 3 \\ ) & \text{if } \mathsf{lr}[i] = l \text{ and } E[i] = 0 \\ \{ & \text{if } \mathsf{lr}[i] = l \text{ and } E[i] = 3 \\ \} & \text{if } \mathsf{lr}[i] = r \text{ and } E[i] = 0 \\ (\} & \text{if } \mathsf{lr}[i] = r \text{ and } E[i] = 1 \\ \{) & \text{if } \mathsf{lr}[i] = l \text{ and } E[i] = 2 \\ [] & \text{if (1) } \mathsf{lr}[i] = l \text{ and } E[i] = 1, \text{ or (2) } \mathsf{lr}[i] = r \text{ and } E[i] = 2 \end{cases}$$

We store a precomputed table that has all possible pairs of lr and E of size $(\log n)/4$ as indices. For each index of the table, it returns lrp constructed from the corresponding pair of lr and E. Thus, the size of the precomputed table is $O(2^{\frac{3}{4}\log n} \log n) = o(n)$ bits

Additionally, we define an imaginary binary sequence $U \in \{1,2\}^{n-1}$ of size $n-1$, where $U[i]$ denotes the number of symbols appended to lrp during its construction by scanning $\mathsf{lr}[i]$ and $E[i]$. Then by Lemma 3, we can decode any $O(\log n)$-sized substring of U starting from position $U[i]$ by storing another precomputed table of size $o(n)$ bits, indexed by all possible pairs of lr and E of size $(\log n)/4$. Consequently, we can support both rank and select queries on U by storing $o(n)$-bit auxiliary structures, without storing U explicitly [3].

To decode any $O(\log n)$-sized substring of lrp starting from position $\mathsf{lrp}[i]$, we first decode a $O(\log n)$-sized substring of E and lr from the position $i' = i - \mathsf{rank}_U(2, i)$ and decode the substring of lrp by accessing the precomputed table a constant number of times (bounded conditions can be easily verified using $\mathsf{rank}_U(2, i-1)$). Thus, without maintaining lrp, we can support rank, select, findopen, and findclose queries on lrp in $O(1)$ time by storing $o(n)$-bit auxiliary structures [3, 10]. With the information provided by lrp and U, we can compute $\mathsf{findopen}(i)$ and $\mathsf{findclose}(i)$ operations on lp in $O(1)$ time as follows: To compute $\mathsf{findopen}(i)$, we compute $i_1 - \mathsf{rank}_U(2, i_1 - 1)$, where i_1 is the position of the matching "(" corresponding to $\mathsf{lrp}[i + \mathsf{rank}_U(2, i)]$. For computing $\mathsf{findclose}(i)$, we similarly compute $i_2 - \mathsf{rank}_U(2, i_2)$, where i_2 corresponds to the position of the ")" corresponding to $\mathsf{lrp}[i + \mathsf{rank}_U(2, i-1)]$. Likewise, we can compute $\mathsf{findopen}(i)$ and $\mathsf{findclose}(i)$ operations on rp by locating the matching "{" or "}" in lrp. In summary, our representation enables findopen and findclose operations on both lp and rp to be supported in $O(1)$ time without storing them explicitly.

Now we show that our representation is valid, i.e., we can decode π from the representation.

▶ **Theorem 4.** *The strings lr and E give a $(3n + o(n))$-bit representation for the Baxter permutation $\pi = (\pi(1), \ldots, \pi(n))$ of size n.*

Proof. It is enough to show that the representation can decode $\mathsf{MinC}(\pi)$ along with the associated labels. For each non-root node $\phi(i)$, we can check $\phi(i)$ is either a left or right child of its parent by referring $\mathsf{lr}[i-1]$. Thus, it is enough to show that the representation can decode the label of the parent of $\phi(i)$. Without loss of generality, suppose $\phi(i)$ is a left child of its parent (the case that $\phi(i)$ is a right child of its parent is analogous). Utilizing the two-stack based algorithm and referring to Lemma 3, we can proceed as follows: If no element is removed from the L stack after traversing $\phi(i-1)$ (this can be checked by referring $\mathsf{lr}[i]$ and $E[i]$), we can conclude that the parent node of $\phi(i)$ is indeed $\phi(i-1)$. Otherwise, the parent of $\phi(i)$ is the node labeled with $\mathsf{findopen}(i-1)$ on lp from the two-stack based algorithm. ◀

▶ **Example 5.** Figure 2 shows the representation of the Baxter permutation $\pi = (9, 8, 10, 1, 7, 4, 5, 6, 2, 3, 11)$. Using the representation, we can access $\mathsf{lp}[3]$ =")" by referring $\mathsf{lr}[3] = l$ and $E[3] = 2$ by Lemma 3. Also, $\mathsf{findopen}(6)$ on lp computed by (1) computing the position of the matching "(" of the parenthesis of lrp at the position $i' = 6 + \mathsf{rank}_U(2, 6) = 8$, which is 5, and (2) returning $5 - \mathsf{rank}_U(2, 5 - 1) = 4$. Note that lrp is not explicitly stored. Finally, we can decode the label of the parent of $\phi(4)$ using $\mathsf{findopen}(3)$ on lp ($\phi(4)$ is the left child of its parent since $\mathsf{lr}[3] = l$), resulting in the value 2. Thus, $\phi(2)$ is the parent of $\phi(4)$.

Representation of alternating Baxter Permutation. Assuming π is an alternating permutation of size n, one can ensure that $\mathsf{MinC}(\pi)$ always forms a full binary tree by introducing, at most, two dummy elements $n + 1$ and $n + 2$, and adding them to the leftmost and rightmost positions of π, respectively [11, 12]. Specifically, we add the node $\phi(i + 1)$ as the leftmost leaf of $\mathsf{MinC}(\pi)$ if $\pi(1) < \pi(2)$, Similarly, we add the node $\phi(i + 2)$ as the rightmost leaf of $\mathsf{MinC}(\pi)$ if $\pi(n - 1) > \pi(n)$.

Since no node in $\mathsf{MinC}(\pi)$ has exactly one child in this case, we can optimize the string E in the representation of Theorem 4 into a binary sequence of size at most $n - 1$, where $E[i]$ indicates whether the node $\phi(i)$ is a leaf node or not. Thus, we can store π using at most $2n + o(n)$ bits. We summarize the result in the following corollary.

▶ **Corollary 6.** *The strings lr and E give a $(2n + o(n))$-bit representation for the alternating Baxter permutation $\pi = (\pi(1), \ldots, \pi(n))$ of size n.*

4 Computing the BP sequence of Cartesian trees

Let π be a Baxter permutation of size n. In this section, we describe how to to compute $\pi(i)$ and $\pi^{-1}(j)$ for $i, j \in \{1, 2, \ldots, n\}$ using the representation of Theorem 4. First in Section 4.1 we modify Cartesian trees so that inorders are assigned to all the nodes. Then we show in Section 4.2 we can obtain the BP sequence of $\mathsf{MinC}(\pi)$ from our representation. By storing the auxiliary data structure of [30], we can support tree navigational operations in Section 2. Finally, in Section 4.4, we show that our data structure can also support the tree navigational queries on $\mathsf{MaxC}(\pi)$ efficiently, which used in the results in the succinct representations of mosaic floorplans and plane bipolar orientations.

To begin discussing how to support $\pi(i)$ and $\pi^{-1}(j)$ queries, we will first show that the representation of Theorem 4 can efficiently perform a depth-first traversal on $\mathsf{MinC}(\pi)$ using its labels. We will establish this by proving the following lemma, which shows that three key operations, namely (1) $\mathsf{left\_child\_label}(i)$: returns the label of the left child of $\phi(i)$, (2) $\mathsf{right\_child\_label}(i)$: returns the label of the right child of $\phi(i)$, and (3) $\mathsf{parent\_label}(i)$: returns the label of the parent of $\phi(i)$ on $\mathsf{MinC}(\pi)$, can be supported in $O(1)$ time.

▶ **Lemma 7.** *The representation of Theorem 4 can support $\mathsf{left\_child\_label}(i)$, $\mathsf{right\_child\_label}(i)$, and $\mathsf{parent\_label}(i)$ in $O(1)$ on $\mathsf{MinC}(\pi)$ in $O(1)$ time.*

Proof. The proof of Theorem 4 shows how to support $\mathsf{parent\_label}(i)$ in $O(1)$ time. Next, to compute $\mathsf{left\_child\_label}(i)$, it is enough to consider the following two cases according to Lemma 3: (1) If $\mathsf{lr}[i] = l$ and $\mathsf{lp}[i]$ is undefined, $\mathsf{left\_child\_label}(i)$ is $i + 1$, and (2) if $\mathsf{lr}[i] = r$ and $\mathsf{lp}[i]$ = '(', we can compute $\mathsf{left\_child\_label}(i)$ in $O(1)$ time by returning $\mathsf{findclose}(i)$ on lp. Similarly, $\mathsf{right\_child\_label}(i)$ can be computed in $O(1)$ time using lr and rp analogously. ◀

Now we can compute $\phi(i + 1)$ from $\phi(i)$ without using the two stacks in $O(1)$ time. We denote this operation by $\mathsf{next}(i)$.

1. If $\mathsf{lr}[i] = l$ and the left child of $\phi(i)$ exists, $\mathsf{next}(i)$ is the left child of $\phi(i)$.
2. If $\mathsf{lr}[i] = r$ and the right child of $\phi(i)$ exists, $\mathsf{next}(i)$ is the right child of v.
3. If $\mathsf{lr}[i] = l$ and the left child of $\phi(i)$ does not exist, $\mathsf{next}(i)$ is the left child of $\phi(j)$ where $j = \mathsf{findclose}(i)$ on lp.
4. If $\mathsf{lr}[i] = r$ and the right child of $\phi(i)$ does not exist, $\mathsf{next}(i)$ is the left child of $\phi(j)$ where $j = \mathsf{findclose}(i)$ on rp.

4.1 Computing inorders

First, we define the inorder of a node in a binary tree. Inorders of nodes are defined recursively as follows. We first traverse the left subtree of the root node and give inorders to the nodes in it, then give the inorder to the root, and finally traverse the right subtree of the root node and give inorders. In [30], inorders are defined for only nodes with two or more children. To apply their data structures to our problem, we modify a binary tree as follows. For each leaf, we add two dummy children. If a node has only right child, we add a dummy left child. If a node has only left child, we add a dummy right child. Then in the BP sequence B of the modified tree, i-th occurrence of ")(" corresponds to the node with inorder i. Therefore we can compute rank and select on ")(" in constant time using the data structure of [30] if we store the BP sequence B of the modified tree explicitly. However, if we do so, we cannot achieve a succinct representation of a Baxter permutation. We implicitly store B. The details are explained next.

4.2 Implicitly storing BP sequences

We first construct B for $\mathsf{MinC}(\pi)$ and auxiliary data structures of [30] for tree navigational operations. In their data structures, B is partitioned into blocks of length ℓ for some parameter ℓ, and search trees called *range min-max trees* are constructed on them. In the original data structure, blocks are stored explicitly, whereas in our data structure, they are not explicitly stored and temporarily computed from our representation. If we change the original search algorithm so that an access to an explicitly stored block is replaced with decoding the block from our representation, we can use the range min-max trees as a black box, and any tree navigational operation works using their data structure. Because the original algorithms have constant query time, they do a constant number of accesses to blocks. If we can decode a block in t time, A tree navigational operation is done in $O(t)$ time. Therefore what remains is, given a position of B, to extract a block of ℓ bits.

Given the inorder of a node, we can compute its label as follows. For each block, we store the following. For the first bit of the block, there are four cases: (1) it belongs to a node in the Cartesian tree. (2) it belongs to two dummy children for a leaf in the Cartesian tree. (3) it belongs to the dummy left child of a node. (4) it belongs to the dummy right child of a node. We store two bits to distinguish these cases. For case (1), we store the label and the inorder of the node using $\log n$ bits, and the information that the parenthesis is either open or close using 1 bit. For case (2), we store the label and the inorder of the parent of the two dummy children, and the offset in the pattern "(()())" of the first bit in the block. For cases (3) and (4), we store the label and the inorder of the parent of the dummy node and the offset in the pattern "()".

To extract a block, we first obtain the label of the first non-dummy node in the block. Then from that node, we do a depth-first traversal using $\mathsf{left\_child\_label}(i)$, $\mathsf{right\_child\_label}(i)$, and $\mathsf{parent\_label}(i)$, and compute a sub-sequence of B for the block. During the traversal, we also recover other dummy nodes. Because the sub-sequence is of length ℓ, there are $O(\ell)$

nodes and it takes $O(\ell)$ time to recover the block. To compute an inorder rank and select, we use a constant number of blocks. Therefore it takes $O(\ell)$ time. The space complexity for additional data structure is $O(n \log n/\ell)$ bits. If we choose $\ell = \omega(\log n)$, the space is $o(n)$.

To support other tree operations including RMin, NSV, and PSV queries on π, we use the original auxiliary data structures of [30]. The space complexity is also $O(n \log n/\ell)$ bits.

4.3 Converting labels and inorders

For the minimum Cartesian tree $\mathsf{MinC}(\pi)$ of Baxter permutation π, the label of the node with inorder i is $\pi(i)$. The inorder of the node with label j is denoted by $\pi^{-1}(j)$.

We showed how to compute the label of the node with given inorder i above. This corresponds to computing $\pi(i)$. Next we consider given label j, to compute the inorder $i = \pi^{-1}(j)$ of the node with label j. Note that $\pi(i) = j$ and $\pi^{-1}(j) = i$ hold.

We use $\mathsf{next}(\cdot)$ to compute the inorder of the node with label j. Assume $i\ell+1 \leq j < (i+1)\ell$. We start from the node $\phi(i\ell + 1)$ with label $i\ell + 1$ and iteratively compute $\mathsf{next}(\cdot)$ until we reach the node with label j. Therefore for $i = 0, 1, \ldots, n/\ell$, we store the positions in the modified BP sequence for nodes $\phi(i\ell + 1)$ using $O(n \log n/\ell)$ bits. If $\mathsf{next}(i\ell + k)$ is a child of $\phi(i\ell + k)$, we can compute its position in the modified BP sequence using the data structure of [30]. If $\mathsf{next}(i\ell + k)$ is not a child of $\phi(i\ell + k)$, we first compute $p = \mathsf{findclose}(i\ell + k)$ on lp or rp. A problem is how to compute the node $\phi(p)$ and its inorder. To compute the inorder of $\phi(p)$, we use *pioneers* of the BP sequence [19]. A pioneer is an open or close parenthesis whose matching parenthesis belongs to a different block. If there are multiple pioneers between two blocks, only the outermost one is a pioneer. The number of pioneers is $O(n/\ell)$ where ℓ is the block size. For each pioneer, we store its position in the BP sequence. Therefore the additional space is $O(n \log n/\ell)$ bits. Consider the case we obtained $p = \mathsf{findclose}(v)$. If v is a pioneer, the inorder of $\phi(p)$ is stored. If v is not a pioneer, we go to the pioneer that tightly encloses v and $\phi(p)$, obtain its position in the BP sequence, and climb the tree to $\phi(p)$. Because $\phi(p)$ and the pioneer belong to the same block, this takes $O(\ell)$ time. Computing a child also takes $O(\ell)$ time. We repeat this $O(\ell)$ times until we reach $\phi(j)$. Therefore the time complexity for converting the label of a node to its inorder takes $O(\ell^2)$ time. The results are summarized as follows.

▶ **Theorem 8.** *For a Baxter permutation π of size n, $\pi(i)$ and $\pi^{-1}(j)$ can be computed in $O(\ell)$ time and $O(\ell^2)$ time, respectively, using a $3n + O(n \log n/\ell)$ bit data structure. This is a succinct representation of a Baxter permutation if $\ell = \omega(\log n)$. The data structure also can support the tree navigational queries in Section 2 on $\mathsf{MinC}(\pi)$, RMin, PSV, and NSV queries in $O(\ell)$ time.*

Note that Theorem 8 also implies that we can obtain the $(2n + o(n))$-bit succinct data structure of an alternating Baxter permutation of size n that support $\pi(i)$ and $\pi^{-1}(j)$ can be computed in $O(\ell)$ time and $O(\ell^2)$ time, respectively, for any $\ell = \omega(\log n)$.

4.4 Navigation queries on Maximum Cartesian trees

In this section, we show the representation of Theorem 4 can also support the tree navigational queries on $\mathsf{MaxC}(\pi)$ in the same time as queries on $\mathsf{MinC}(\pi)$, which will be used in the succinct representations of mosaic floorplans and plane bipolar orientations.

Note that we can traverse the nodes in $\mathsf{MaxC}(\pi)$ according to the decreasing order of their labels, using the same two-stack based algorithm as described in Section 3. Now, let $\phi'(i)$ represent the node in $\mathsf{MaxC}(\pi)$ labeled with i. We then define sequences lr and E

on $\mathsf{MaxC}(\pi)$ in a manner analogous to the previous definition (we denote them as lr', and E', respectively). The only difference is that the value of i-th position of these sequences corresponds to the node $\phi'(n-i+1)$ instead of $\phi(i)$, since we are traversing from the node with the largest label while traversing $\mathsf{MaxC}(\pi)$. Then by Theorem 4 and 8, it is enough to show how to decode any $O(\log n)$-size substring of lr' and E' from lr and E, respectively.

We begin by demonstrating that for any $i \in [1,\dots,n-1]$, the value of $\mathsf{lr}'[i]$ is l if and only if $\mathsf{lr}[n-i]$ is r. As a result, our representation can decode any $O(\log n)$-sized substring of lr' in constant $O(1)$ time. Consider the case where $\mathsf{lr}[i]$ is l (the case when $\mathsf{lr}[i]=r$ is handled similarly). In this case, according to the two-stack based algorithm, $\phi(i+1)$ is the left child of $\phi(i_1)$, where $i_1 \leq i$. Now, we claim that $\phi'(i)$ is the right child of its parent. Suppose, for the sake of contradiction, that $\phi'(i)$ is a left child of $\phi'(i_2)$. Then $\phi(i+1)$ cannot be an ancestor of $\phi(i)$, as there are no labels between $i+1$ and i. Thus, $i_2 > i+1$, and there must exist a lowest common ancestor of $\phi'(i)$ and $\phi'(i+1)$ (denoted as $\phi'(k)$). At this point, $\phi'(i+1)$ and $\phi'(i_2)$ reside in the left and right subtrees rooted at $\phi'(k)$, respectively. Now $i_3 \leq i$ be a leftmost leaf of the subtree rooted at $\phi'(i)$. Then there exists a pattern 2–41—3 induced by $(i+1)-k$ and i_3-i_2, which contradicts the fact that π is a Baxter permutation.

Next, we show that the following lemma implies that the representation can also decode any $O(\log n)$-size substring of E' in $O(1)$ time from E along with $\pi(1)$ and $\pi(n)$.

▶ **Lemma 9.** *Given a permutation π, $\phi(i)$ has a left child if and only if $\pi^{-1}(i) > 1$ and $\pi(\pi^{-1}(i)-1) > i$. Similarly, $\phi(i)$ has a right child if and only if $\pi^{-1}(i) < n$ and $\pi(\pi^{-1}(i)+1) > i$.*

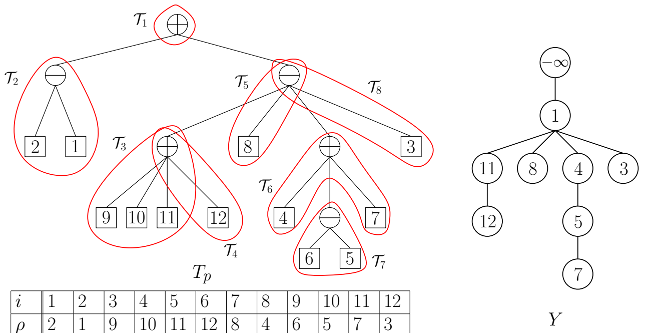
Proof. We only prove that $\phi(i)$ has a left child if and only if $\pi^{-1}(i) > 1$ and $\pi(\pi^{-1}(i)-1) < i$ (the other statement can be proved using the same argument). Let i_1 be $\pi(\pi^{-1}(i)-1)$. From the definition of the minimum Cartesian tree, if $\phi(i_1)$ is at the left subtree of $\phi(i)$, it is clear that $i_1 > i$. Now, suppose $i_1 > i$, but $\phi(i)$ does not have a left child. In this case, $\phi(i)$ cannot be an ancestor of $\phi(i_1)$. Thus, there must exist an element in π positioned between i_1 and i, which contradicts the fact that they are consecutive elements. ◀

As a conclusion, the data structure of Theorem 8 can support the tree navigational queries in Section 2 on $\mathsf{MaxC}(\pi)$, and RMax, PSV, and NSV queries in $\omega(\log n)$ time using $o(n)$-bit auxiliary structures from the results in Section 4.2. We summarize the results in the following theorem.

▶ **Theorem 10.** *For a Baxter permutation π of size n, The succinct data structure of Theorem 8 on π can support the tree navigational queries in Section 2 on $\mathsf{MaxC}(\pi)$, RMax, PLV, and NLV queries in $O(f_1(n))$ time for any $f_1(n) = \omega(\log n)$.*

5 Succinct Data Structure of Separable Permutation

In this section, we present a succinct data structure for a separable permutation $\rho = (\rho(1),\dots,\rho(n))$ of size n that supports $\rho(i)$ and $\rho^{-1}(j)$ in $O(1)$ time. The main idea of the data structure is as follows. It is known that for the separable permutation ρ, there exists a unique *separable tree ($v-h$ tree)* T_ρ of n leaves [7,33], which will be defined later. Since T_ρ is a labeled tree with at most $2n-1$ nodes, $O(n\log n)$ bits are necessary to store T_ρ explicitly. Instead, we store it using a tree covering where each micro-tree of T_ρ is stored as an index of the precomputed table that maintains all separable permutations whose separable trees have at most ℓ_2 nodes, where ℓ_2 is a parameter of the size of the micro-tree of T_ρ, which will be decided later. After that, we show how to support the queries in Theorem 8 and 10 in $O(1)$ time using the representation, along with $o(n)$-bit auxiliary structures.



i	1	2	3	4	5	6	7	8	9	10	11	12
ρ	2	1	9	10	11	12	8	4	6	5	7	3
B	1	0	1	1	1	0	1	1	1	0	0	1

$$S_\rho = 1 \quad 9 \quad 12 \quad 8 \quad 4 \quad 5 \quad 7 \quad 3$$
$$\rho' = 1 \quad 11 \quad 12 \quad 8 \quad 4 \quad 5 \quad 7 \quad 3$$

Figure 3 An example of the representation of the separable permutation $\rho = (2, 1, 9, 10, 11, 12, 8, 4, 6, 5, 7, 3)$. Each tree within the red area represents a mini-tree of T_ρ with $\ell_1 = 3$.

5.1 Succinct Representation

Given a separable permutation ρ of size n, the separable tree T_ρ of ρ is an ordered tree with n leaves defined as follows [33]:

- Each non-leaf node of T_ρ is labeled either $\oplus$ or $\ominus$. We call a $\oplus$ node as an internal node labeled with $\oplus$, and similarly, a $\ominus$ node as an internal node labeled with $\ominus$.
- The leaf node of T_ρ whose leaf rank (i.e., the number of leaves to the left) i has a label $\rho(i)$. In the rest of this section, we refer to it as the leaf $\rho(i)$.
- Any non-leaf child of $\oplus$ node is a $\ominus$ node. Similarly, any non-leaf child of $\ominus$ node is a $\oplus$ node.
- For any internal node $p \in T_\rho$, let ρ_p be a sequence of the labels of p's children from left to right, as replacing the label of non-leaf child of p to the label of the leftmost leaf node in the rooted subtree at the node. Then if p is a $\oplus$ (resp. $\ominus$ node), ρ_p is an increasing (resp. decreasing) subsequence of ρ.

See Figure 3 for an example. Szepienic and Otten [33] showed that for any separable permutation of size n, there exists a unique separable tree of it with n leaves.

We maintain ρ through the tree covering algorithm applied to T_ρ, with the parameters for the sizes of mini-trees and micro-trees as $\ell_1 = \log^2 n$ and $\ell_2 = \frac{\log n}{6}$, respectively. Here, the precomputed table maintains all possible separable permutations whose corresponding separable trees have at most ℓ_2 nodes. Additionally, two special cases are considered: when the micro-tree is a singleton $\oplus$ or $\ominus$ node. Since any separable permutation stored in the precomputed table has a size at most ℓ_2, there exist $o(n)$ indices in the precomputed table.

The micro-trees of T_ρ are stored as their corresponding indices in the precomputed table, using $n \log(3 + 2\sqrt{2}) + o(n) \simeq 2.54n + o(n)$ bits in total. In the full version of the paper [8], we consider how to support $\rho(i)$ and $\rho^{-1}(j)$ in $O(1)$ time. The data structure also supports RMin, RMax, PSV, PLV, NSV, and NLV on ρ in $O(\log\log n)$ time.

6 Future Work

We conclude with the following concrete problems for possible further work in the future: (1) Can we improve the query times of π and π^{-1} for Baxter permutations? (2) can we show any time/space trade-off lower bound for Baxter permutation similar to that of general permutation [20]? and (3) are there any succinct data structures for other pattern-avoiding permutations?

References

1 Eyal Ackerman, Gill Barequet, and Ron Y. Pinter. A bijection between permutations and floorplans, and its applications. *Discret. Appl. Math.*, 154(12):1674–1684, 2006. doi:10.1016/J.DAM.2006.03.018.

2 Glen Baxter. On fixed points of the composite of commuting functions. *Proceedings of the American Mathematical Society*, 15(6):851–855, 1964.

3 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms (TALG)*, 11(4):1–21, 2015. doi:10.1145/2629339.

4 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *CPM*, volume 6129 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2010. doi:10.1007/978-3-642-13509-5_13.

5 Miklós Bóna. *Combinatorics of Permutations, Second Edition.* Discrete mathematics and its applications. CRC Press, 2012.

6 Nicolas Bonichon, Mireille Bousquet-Mélou, and Éric Fusy. Baxter permutations and plane bipolar orientations. *Electron. Notes Discret. Math.*, 31:69–74, 2008. doi:10.1016/j.endm.2008.06.011.

7 Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Inf. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.

8 Sankardeep Chakraborty, Seungbum Jo, Geunho Kim, and Kunihiko Sadakane. Succinct data structures for baxter permutation and related families, 2024. arXiv:2409.16650.

9 Sankardeep Chakraborty, Seungbum Jo, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct data structures for bounded clique-width graphs. *Discret. Appl. Math.*, 352:55–68, 2024. doi:10.1016/J.DAM.2024.03.016.

10 Richie Chih-Nan Chuang, Ashim Garg, Xin He, Ming-Yang Kao, and Hsueh-I Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. In *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*, pages 118–129. Springer, 1998. doi:10.1007/BFB0055046.

11 Robert Cori, Serge Dulucq, and Gérard Viennot. Shuffle of parenthesis systems and baxter permutations. *Journal of Combinatorial Theory, Series A*, 43(1):1–22, 1986. doi:10.1016/0097-3165(86)90018-X.

12 Serge Dulucq and Olivier Guibert. Stack words, standard tableaux and baxter permutations. *Discrete Mathematics*, 157(1-3):91–106, 1996. doi:10.1016/S0012-365X(96)83009-3.

13 Serge Dulucq and Olivier Guibert. Baxter permutations. *Discrete Mathematics*, 180(1-3):143–156, 1998. doi:10.1016/S0012-365X(97)00112-X.

14 A. Farzan and J. I. Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, January 2014. doi:10.1007/S00453-012-9664-0.

15 Stefan Felsner, Éric Fusy, Marc Noy, and David Orden. Bijections for baxter families and related objects. *J. Comb. Theory, Ser. A*, 118(3):993–1020, 2011. doi:10.1016/J.JCTA.2010.03.017.

16 Johannes Fischer. Combined data structure for previous- and next-smaller-values. *Theor. Comput. Sci.*, 412(22):2451–2456, 2011. doi:10.1016/J.TCS.2011.01.036.

17 Éric Fusy. Straight-line drawing of quadrangulations. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing, 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers*, volume 4372 of *Lecture Notes in Computer Science*, pages 234–239. Springer, 2006. doi:10.1007/978-3-540-70904-6_23.

18 Paweł Gawrychowski and Patrick K Nicholson. Optimal encodings for range top-k k, selection, and min-max. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I 42*, pages 593–604. Springer, 2015. doi:10.1007/978-3-662-47672-7_48.

19 Richard F. Geary, Naila Rahman, Rajeev Raman, and Venkatesh Raman. A simple optimal representation for balanced parentheses. *Theoretical Computer Science*, 368(3):231–246, 2006. Combinatorial Pattern Matching. doi:10.1016/J.TCS.2006.09.014.

20 Alexander Golynski. Cell probe lower bounds for succinct data structures. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 625–634. SIAM, 2009. doi:10.1137/1.9781611973068.69.

21 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

22 Bryan Dawei He. A simple optimal binary representation of mosaic floorplans and baxter permutations. *Theoretical Computer Science*, 532:40–50, 2014. doi:10.1016/J.TCS.2013.05.007.

23 Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, Chung-Kuan Cheng, and Jun Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140)*, pages 8–12. IEEE, 2000. doi:10.1109/ICCAD.2000.896442.

24 Seungbum Jo and Geunho Kim. Space-efficient data structure for next/previous larger/smaller value queries. In *LATIN*, volume 13568 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2022. doi:10.1007/978-3-031-20624-5_5.

25 Thomas Lengauer. *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.

26 J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

27 J Ian Munro, Rajeev Raman, Venkatesh Raman, et al. Succinct representations of permutations and functions. *Theoretical Computer Science*, 438:74–88, 2012. doi:10.1016/J.TCS.2012.03.005.

28 J Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

29 Masahiro Nakano, Akisato Kimura, Takeshi Yamada, and Naonori Ueda. Baxter permutation process. *Advances in Neural Information Processing Systems*, 33:8648–8659, 2020.

30 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms (TALG)*, 10(3):1–39, 2014. doi:10.1145/2601073.

31 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4):43–es, 2007.

32 Zion Cien Shen and Chris CN Chu. Bounds on the number of slicing, mosaic, and general floorplans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(10):1354–1361, 2003. doi:10.1109/TCAD.2003.818136.

33 Antoni A. Szepieniec and Ralph H. J. M. Otten. The genealogical approach to the layout problem. In *DAC*, pages 535–542. ACM/IEEE, 1980. doi:10.1145/800139.804582.

34 Roberto Tamassia and Ioannis G. Tollis. A unified approach a visibility representation of planar graphs. *Discret. Comput. Geom.*, 1:321–341, 1986. doi:10.1007/BF02187705.

35 Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

36 Bo Yao, Hongyu Chen, Chung-Kuan Cheng, and Ronald L. Graham. Floorplan representations: Complexity and connections. *ACM Trans. Design Autom. Electr. Syst.*, 8(1):55–80, 2003. doi:10.1145/606603.606607.

37 Xiaoke Zhu, Changwen Zhuang, and Y. Kajitani. A general packing algorithm based on single-sequence. In *2004 International Conference on Communications, Circuits and Systems*, volume 2, pages 1257–1261 Vol.2, July 2004. doi:10.1109/ICCCAS.2004.1346402.

38 Changwen Zhuang, Xiaoke Zhu, Y. Takashima, S. Nakatake, and Y. Kajitani. An algorithm for checking slicing floorplan based on hpg and its application. In *2004 International Conference on Communications, Circuits and Systems (IEEE Cat. No.04EX914)*, volume 2, pages 1223–1227 Vol.2, 2004. doi:10.1109/ICCCAS.2004.1346395.

Enhancing Generalized Compressed Suffix Trees, with Applications

Sankardeep Chakraborty
The University of Tokyo, Japan

Kunihiko Sadakane
The University of Tokyo, Japan

Wiktor Zuba
CWI, Amsterdam, The Netherlands

Abstract

Generalized suffix trees are data structures for storing and searching a set of strings. Though many string problems can be solved efficiently using them, their space usage can be large relative to the size of the input strings. For a set of strings with n characters in total, generalized suffix trees use $O(n \log n)$ bit space, which is much larger than the strings that occupy $n \log \sigma$ bits where σ is the alphabet size. Generalized compressed suffix trees use just $O(n \log \sigma)$ bits but support the same basic operations as the generalized suffix trees. However, for some sophisticated operations we need to add auxiliary data structures of $O(n \log n)$ bits. This becomes a bottleneck for applications involving big data. In this paper, we enhance the generalized compressed suffix trees while still retaining their space efficiency. First, we give an auxiliary data structure of $O(n)$ bits for generalized compressed suffix trees such that given a suffix s of a string and another string t, we can find the suffix of t that is closest to s. Next, we give a $o(n)$ bit data structure for finding the ancestor of a node in a (generalized) compressed suffix tree with given string depth. Finally, we give data structures for a generalization of the document listing problem from arrays to trees. We also show their applications to suffix-prefix matching problems.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases suffix tree, compact data structure, suffix-prefix query, weighted level ancestor

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.18

Funding *Sankardeep Chakraborty*: Supported by MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant Number JPMXS0120319794.
Wiktor Zuba: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 101034253 and 872539.

1 Introduction

Suffix trees are data structures for string matching [29]. In addition to the basic pattern matching problem, they can also be used for other problems such as finding longest common extensions, maximal pairs, approximate string matching, etc. [14]. They can be further extended to generalized suffix trees (GSTs for short) storing suffixes of a set of strings, which gives them many applications in bioinformatics such as longest common substrings, maximal unique matches [5], and maximal exact matches [17]. Hereafter we do not distinguish suffix trees and GSTs unless specified because GST is the suffix tree of the string obtained by concatenating all the strings from the set.

Though suffix trees are the most basic data structures in string processing, one drawback is their space usage. Though the suffix tree of a string uses $O(n)$ machine words, where n is the string length, that alone already requires huge memory. It was estimated that for a human genome, which has about 3 billion characters, the suffix tree uses more than 40 GB of memory [16]. Therefore there has been much research on reducing the space

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 18; pp. 18:1–18:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

requirement of suffix trees. There are two approaches; one is to omit some of the components of suffix trees and the other is to compress the components. For the first approach, suffix arrays [19], enhanced suffix arrays [1], and space efficient suffix trees [21] have been proposed as space-efficient alternatives to the suffix trees. For the second approach, the compressed suffix arrays [13] and compressed suffix trees [25] have been proposed for compressing the respective standard structures. The first approach aims mainly at reduction of the practical space usage as asymptotically the space usage remains the same; suffix arrays and enhanced suffix arrays use $\mathrm{O}(n \log n)$ bits of space for a string of length n – the same as the suffix trees. The second approach, on the other hand, aims at improving the asymptotic bounds; compressed suffix arrays and trees use $\mathrm{O}(n \log \sigma)$ bits where σ is the size of the alphabet of the string. These data structures are truly linear space data structures – the space is linear to the actual input size – $n \log \sigma$ bits[1] (the space needed to represent the string).

Though the compressed suffix trees support the same set of basic operations as the suffix trees, some of auxiliary data structures for supporting extended operations still use $\mathrm{O}(n \log n)$ bit space, which dominates the space of the entire data structure.

1.1 Our contributions

In this paper, we enhance the generalized compressed suffix trees. First we add auxiliary data structures which for a given suffix t and a string ID i allow finding the suffix of the string i most similar to t. For two given suffixes it is easy to compute the length of their longest common prefix using the suffix tree even if the suffixes belong to different strings. However, if a suffix t of a string is fixed and the ID i of another string is given, finding the suffix of the string i with the longest common prefix with s takes time due to the multiplicity of the possible candidate suffixes.

▶ **Theorem 1** (Closest colored suffixes). *We are given a set of strings $S_1, S_2, \ldots, S_k$ on an alphabet of size σ. The total length of the strings is n. There exists a data structure using* $\mathrm{SIZE}_{\mathrm{SA}}(2n, \sigma) + \mathrm{O}(n + k)$ *bits so that given a suffix t of a string S_j and an index i, we can obtain the suffix s of S_i that has the maximum LCP with t in* $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log k)$ *time, where* $\mathrm{SIZE}_{\mathrm{SA}}(n, \sigma)$ *is the size of a data structure storing a suffix array for a string of length n on an alphabet of size σ, and* $\mathrm{TIME}_{\mathrm{SA}}$ *is the time for obtaining an entry of a suffix array or its inverse.*

Note that $\mathrm{TIME}_{\mathrm{SA}}$ also depends on n and σ in general, but we omit them because they are fixed throughout the paper. If we use the data structure in the second row of Table 1, the space and the time complexities become $2n \log \sigma + \mathrm{O}(n + k)$ bits and $\mathrm{O}(\log n \log \log k)$ time, respectively. An existing solution [18] has $\mathrm{O}(\log \log k)$ query time using $\mathrm{O}(n \log n)$ bit space. Our algorithm is faster than the original GST, which is $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log n)$ time.

Next we give a succinct index for weighted level ancestors in compressed suffix trees.

▶ **Theorem 2** (Weighted level ancestors). *By adding an auxiliary data structure of* $\mathrm{o}(n)$ *bits to the compressed suffix tree, we can compute the nearest ancestor of a node with string depth smaller than a given value in* $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$ *time.*

This is faster than the original GST, which is $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log n)$ time. The proofs are given in Section 3.

[1] Throughout the paper the base of the logarithm is two.

Table 1 Size in bits and query time of suffix arrays and compressed suffix arrays, where n is the length of the string and σ is its alphabet size. H_k is the k-th order entropy of the string. Time for obtaining an entry of the suffix array is denoted by $\mathrm{TIME}_{\mathrm{SA}}$.

Index	Space ($\mathrm{SIZE}_{\mathrm{SA}}(n,\sigma)$)	Query time ($\mathrm{TIME}_{\mathrm{SA}}$)
Suffix array [19]	$n \log n$	$\mathrm{O}(1)$
Compressed suffix array [13]	$n \log \sigma + \mathrm{O}(n)$	$\mathrm{O}(\log n)$
Compressed suffix array [13]	$\mathrm{O}(\epsilon^{-1} n \log \sigma)$	$\mathrm{O}(\log^{\epsilon} n)$
FM-index [7]	$nH_k + \mathrm{o}(n \log \sigma)$	$\mathrm{O}(\frac{\log \sigma}{\log \log n})$

We also give applications of these two enhancements in Section 4. Our proposed data structures are used to solve the suffix-prefix matching problems [18] (see also [31] for their approximate version). Existing solutions use $\mathrm{O}(n \log n)$ bit space, whereas ours use linear ($\mathrm{O}(n \log \sigma)$ bit) space. For solving this problem, we generalize the document listing problem [22] from arrays to trees providing data structures that are of independent interest.

2 Preliminaries

2.1 Suffix arrays, suffix trees, and their compression

A string S of length n on an alphabet $\mathcal{A}$ is an array $S[1,n]$ of characters in $\mathcal{A}$. We assume the alphabet is an ordered set. We add a terminator \$ at the end of the string, that is, $S[n{+}1] = \$$, which is smaller than any character in $\mathcal{A}$. The character at position i in the string S is denoted by $S[i]$. A substring of S is the concatenation of characters $S[i], S[i{+}1], \ldots, S[j]$ and denoted by $S[i,j]$. Substrings of the form $S[i,n]$ and $S[1,i]$ are called suffixes and prefixes, respectively. For two strings s,t, $\mathrm{LCP}(s,t)$ is defined to be the length of the longest common prefix between them.

The suffix array [19] of a string S of length n is an integer array $\mathrm{SA}[0,n]$, where $\mathrm{SA}[i] = j$ means that the suffix $S[j,n]$ is lexicographically the i-th suffix among all the suffixes of S ($S[0] = n+1$). The (classic) suffix array uses $n \log n$ bits of space for the array SA, and $n \log \sigma$ bits of space for the string S itself.

The suffix tree of a string is a compacted trie representing all the suffixes of the string [29]. The suffix tree has $n+1$ leaves, each corresponding to a suffix of S (including the last suffix $S[n+1,n+1] = \$$). Each edge of the suffix tree has a string label. We define the string label of a node as the concatenation of the labels of the edges between the root and the node. The string label of the i-th leaf coincides with lexicographically the i-th suffix. The string depth of an internal node v of the suffix tree is the length of the string label of v. For two suffixes s and t, $\mathrm{LCP}(s,t)$ is equal to the string depth of the lowest common ancestor between their corresponding leaves.

The suffix array can be compressed to $\mathrm{O}(n \log \sigma)$ bits where $\sigma = |\mathcal{A}|$ so that each entry $\mathrm{SA}[i]$ can be computed in $\mathrm{polylog}(n)$ time [13]. The inverse suffix array $\mathrm{ISA}[1,n+1]$ of a string is an integer array such that $\mathrm{ISA}[j] = i$ if and only if $\mathrm{SA}[i] = j$. The inverse array can be computed within the same time complexity as the suffix array. Let $\mathrm{TIME}_{\mathrm{SA}}$ denote the time for computing a value of a suffix array or an inverse suffix array. The space and query time complexities for compressed versions of the suffix arrays are shown in Table 1.

The compressed suffix tree [25] of a string S consists of the compressed suffix array of S, a balanced parentheses (BP) representation [20, 23] of the compacted trie, and a bit-vector storing the information about the string depths of nodes. The second and the third components use $4n + \mathrm{o}(n)$ bits and $2n + \mathrm{o}(n)$ bits, respectively. Using these components, we can compute the string depth of a node in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time.

Using the compressed suffix tree, we can support the following operations:

- Finding the leaf corresponding to the lexicographically i-th suffix in constant time.
- Finding the lowest common ancestor of two nodes in constant time.
- Finding the level ancestor (the ancestor of a node with given depth) in constant time [23]. Note that this depth is not the string depth.
- Computing the string depth of a node in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time.
- Computing the edge labels of length ℓ in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} + \ell)$ time.

Note that the compressed suffix tree of [25] supports weighted level ancestor queries w.r.t. string depths in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log n)$ time by a binary search using (unweighted) level ancestor queries and string depth queries. If we can use $\mathrm{O}(n \log n)$ bits of space we can support this query in constant time [3]. In this paper we give an index supporting the queries in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log n)$ time using additional $\mathrm{o}(n)$ bits to the compressed suffix trees.

2.2 Bit-vectors and rank/select dictionaries

A bit-vector is a string $B[1, n]$ on alphabet $\{0, 1\}$ supporting the following three operations.

- $\mathrm{access}(B, i)$: returns $B[i]$.
- $\mathrm{rank}_c(B, i)$: returns the number of c's ($c \in \{0, 1\}$) in $B[1, i]$.
- $\mathrm{select}_c(B, i)$: returns the position of the i-th occurrence of $c \in \{0, 1\}$. If $i > \mathrm{rank}_c(B, n)$, we define $\mathrm{select}_c(B, i) = n + 1$. We also define $\mathrm{select}_c(B, 0) = 0$.

We can perform each of these operations in constant time using $n + \mathrm{o}(n)$ bits of space [24].

The predecessor $\mathrm{pred}_c(B, i)$ and the successor $\mathrm{succ}_c(B, i)$ are the positions of c closest to i. They can be computed in constant time as $\mathrm{pred}_c(B, i) := \mathrm{select}_c(B, \mathrm{rank}_c(B, i - 1))$ and $\mathrm{succ}_c(B, i) := \mathrm{select}_c(B, \mathrm{rank}_c(B, i) + 1)$. Note that $\mathrm{pred}_c(B, i) < i < \mathrm{succ}_c(B, i)$.

2.3 Generalized suffix arrays and trees

We are given a set of strings $S_1, S_2, \ldots, S_k$ on an alphabet of size σ. We concatenate them into string $\mathcal{S} = S_1\$S_2\$ \cdots S_k\$$. The generalized suffix array/tree of the set of the strings is just the suffix array/tree of $\mathcal{S}$ with the following modification.

We create a bit-vector D of length $n + k$ where $n = |S_1| + |S_2| + \cdots + |S_k|$, and set 1's for the positions of \$'s in $\mathcal{S}$. Then, given a position j in $\mathcal{S}$, we can compute the ID d of the string S_d containing the position j in constant time by $d := \mathrm{rank}_1(D, j) + 1$. We define the document array $A[0, n + k]$ as $A[i] := \mathrm{rank}_1(D, \mathrm{SA}_{\mathcal{S}}[i]) + 1$. we do not store the document array explicitly because it uses $n \log k$ bits and each entry can be computed from the (compressed) suffix array in $\mathrm{TIME}_{\mathrm{SA}}$ time.

Sadakane [26] enhanced generalized compressed suffix trees to compute the number of occurrences of a pattern in each of the strings efficiently. After creating (compressed) suffix arrays (and inverse suffix arrays) of $\mathcal{S}$ and each S_d we can convert the rank r_{l} of a suffix of S_d into the rank r_{g} in $\mathcal{S}$ and vice versa in $\mathrm{O}(t_{SA})$ time as follows.

$$r_{\mathrm{g}} = \mathrm{ISA}_{\mathcal{S}}[\mathrm{SA}_{S_d}[r_{\mathrm{l}}] + s_d] \qquad (1)$$

$$r_{\mathrm{l}} = \mathrm{ISA}_{S_d}[\mathrm{SA}_{\mathcal{S}}[r_{\mathrm{g}}] - s_d] \qquad (2)$$

where $s_d = |S_1| + |S_2| + \cdots + |S_{d-1}| + d - 1$. We can compute s_d in constant time using the bit-vector D, namely, $s_d = \mathrm{select}_1(D, d - 1)$. Recall that d is computed from r_{g} by $d := \mathrm{rank}_1(D, r_{\mathrm{g}}) + 1$. We call r_{g} and r_{l} as the global and the local rank of the suffix, respectively.

Figure 1 shows the generalized suffix tree for a set of strings ACAA, ACAG, ACGC, CACA. Note that we use the same example as [18]. Each string is appended with a terminator \$. To bound the degree of a node by $\sigma + 1$, we add an artificial node if a node has more than

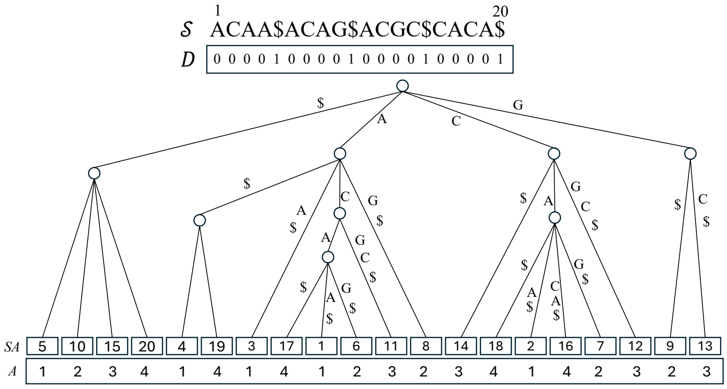

1
20
$\mathcal{S}$ ACAA\$ACAG\$ACGC\$CACA\$
D 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
SA 5 10 15 20 4 19 3 17 1 6 11 8 14 18 2 16 7 12 9 13
A 1 2 3 4 1 4 1 4 1 2 3 2 3 4 1 4 2 3 2 3

Figure 1 Generalized suffix tree for a set of strings ACAA, ACAG, ACGC, CACA. SA is the suffix array of the concatenated string $\mathcal{S}$, and A is the array storing ID's of suffixes.

one edge with label starting from \$. For example, the root node has an edge with label \$ pointing to a node with four edges labeled \$. We can distinguish \$'s by their positions in $\mathcal{S}$. We define the string depth of an artificial node as that of its parent node. We can compute it using the same algorithm as that for normal nodes.

2.4 LCP arrays

For a string T of length n and its suffix array SA, we define the LCP (longest common prefix) array $L[1, n]$ as

$$L[i] = \mathrm{LCP}(T[SA[i-1], n], T[SA[i], n]).$$

If we store L in plain form, we need $n \log n$ bits of space. However, if we have access to the suffix array, we can compress it into $2n + \mathrm{o}(n)$ bits so that any entry of $L[i]$ is computed in $\mathrm{TIME}_{\mathrm{SA}} + \mathrm{O}(1)$ time [25].

The length of the LCP between two suffixes $T[SA[p], n]$ and $T[SA[q], n]$ of a string can be obtained by $\min_{p<i\le q} L[i]$. The index i attaining the minimum value can be computed in constant time using the $2n + \mathrm{o}(n)$ bit data structure for range minimum queries [8]. We can also compute it using the compressed suffix tree [25]. In this case, we use $4n + \mathrm{o}(n)$ bits for the tree topology of the suffix tree.

2.5 Rank and select data structures for large alphabets

Let $T[1, n]$ be a string on alphabet $\mathcal{A}$ of size k. As a generalization of the case of bit-vectors, we can define operations $\mathrm{access}(T, i)$, $\mathrm{rank}_c(T, i)$, and $\mathrm{select}_c(T, i)$ for $c \in \mathcal{A}$. The wavelet tree [12] supports all the operations in $\mathrm{O}(\log k)$ time using $(n+\mathrm{o}(n)) \log k$ bit space. Golynski et al. [11] gave a data structure supporting select in constant time and rank and access in $\mathrm{O}(\log \log k)$ time using $(n + \mathrm{o}(n)) \log k + 2n$ bit space[2].

These data structures encode the string in a specific form. Therefore we cannot further compress the string. Barbay et al. [2] considered another approach; they design succinct indexes for abstract data types. Their results are summarized as follows[3].

[2] There are other variants.
[3] The claim is slightly simplified from the original one.

▶ **Theorem 3** (Theorem 2.10 in [2]). *Given support for* select *in $f(n,k)$ time on a string S of length n on an alphabet of size k, we can support* access, rank*, predecessor, and successor (for any character) in* $\mathrm{O}(f(n,k)\log\log k)$ *time with a succinct index using* $\mathrm{O}(n\log k/\log\log k)$ *bits of space.*

Though this index uses asymptotically smaller space than the string itself, its size still depends on the alphabet size k. In this paper we follow the abstract data type approach and give a new index using less space (see Lemma 6).

2.6 Nearest marked ancestors

Let T be a rooted tree with some of the nodes marked. In the nearest marked ancestor problem, for a given node v of T we want to find its closest marked ancestor. Tsur [28] gave solutions for a generalized version of this problem – nearest colored ancestor – where each node has a color and we find the nearest ancestor with a given color. In this paper we only use the nearest marked ancestor queries, thus we provide a simplified statement.

▶ **Theorem 4** (A simplified version of Theorem 1 of [28]). *There exists a representation of T that uses $n + \mathrm{o}(n)$ bits in addition to a $2n + \mathrm{o}(n)$ bit representation of the tree topology that allows for answering the nearest marked ancestor queries in* $\mathrm{O}(1)$ *time.*

2.7 Weighted level ancestor queries

Consider a rooted tree with n nodes where each node has an integer weight in $[0, U]$ and on any path from a leaf to the root, the weights are non-increasing. The weighted level ancestor $\mathrm{WA}(v, w)$ of node v is the closest node on the path from v to the root that has weight smaller than w. Kopelowitz and Lewenstein [15] gave a data structure using linear space that answers a query in the same time complexity as finding the predecessor among n values in $[0, U]$. For the case of $U = \mathrm{O}(n)$, we obtain a data structure using $\mathrm{O}(n \log n)$ bits of space that answers a query in $\mathrm{O}(\log\log n)$ time. As shown above, for the case that weights are equal to the string depths in a suffix tree, there is a data structure using $\mathrm{O}(n \log n)$ bits of space that supports the query in constant time [3]. In Section 3.3 we show how to reduce the space at the cost of more expensive queries.

2.8 Tree Covering

Here we provide an overview of Farzan and Munro's tree covering representation and its application in creating a succinct tree data structure [6]. Their approach involves decomposing the input tree into smaller units called "mini-trees", which are further divided into "micro-trees." These micro-trees are stored efficiently in a compact precomputed table. By focusing on the connections and links between these subtrees, the entire tree can be represented using the shared roots of the mini-trees. The main result is the following theorem.

▶ **Theorem 5** ([6]). *For a rooted ordered tree with n nodes and a positive integer $1 \le \ell \le n$, one can decompose the trees into subtrees satisfying the following conditions: (1) each subtree contains at most 2ℓ nodes, (2) the number of subtrees is* $\mathrm{O}(n/\ell)$*, (3) each subtree has at most one outgoing edge, apart from those from the root of the subtree.*

Note that to achieve this, we allow subtrees to share their root nodes, hence the name "tree covering". Theorem 5 applied with $\ell = \log^2 n$ creates a tree covering representation for a tree with n nodes, resulting in $\mathrm{O}(n/\log^2 n)$ mini-trees. The resulting *tree over mini-trees*, with $\mathrm{O}(n/\log^2 n)$ nodes, can be represented in $\mathrm{O}(n/\log n) = \mathrm{o}(n)$ bits. Theorem 5 can be then applied to each mini-tree with $\ell_1 = \frac{1}{6}\log n$, resulting in $\mathrm{O}(n/\log n)$ micro-trees. The *mini-tree over micro-trees* is obtained by contracting each micro-tree into a node and adding dummy nodes for micro-trees sharing a common root, and it has $\mathrm{O}(\log n)$ vertices, thus, it can be represented by $\mathrm{O}(\log\log n)$-bit pointers. The total space for all mini-trees over micro-trees is $\mathrm{O}(n\log\log n/\log n) = \mathrm{o}(n)$ bits. Micro-trees are stored with two-level pointers in a precomputed table, which occupies $2n + \mathrm{o}(n)$ bits. This representation, supplemented by auxiliary data structures requiring only $\mathrm{o}(n)$ bits, enables fundamental tree navigation operations, such as accessing the parent, the i-th child, the lowest common ancestor, and many more in $\mathrm{O}(1)$ time [6].

2.9 Document listing problems

The document listing problem [22] is, given an array of colors $A[1,n]$ and an interval $[i,j]$ of the indices of the array, to enumerate all distinct colors in the sub-array $A[i,j]$. The problem can be solved in optimal $\mathrm{O}(1+k)$ time where k is the output size (the number of distinct colors in $A[i,j]$), after $\mathrm{O}(n)$ time preprocessing for A. Namely, the preprocessing first constructs another array $P[1,n]$ such that $P[i] = j$ if $j < i$ is the largest index such that $A[i] = A[j]$, and $P[i] = -1$ if no such j exists, and then constructs a range minimum query data structure for P. We can consider P to be a representation of linked lists connecting the same colors.

A query for $A[i,j]$ is done as follows. First we find the index m of the minimum value in $P[i,j]$. If $P[m] < i$, we output $A[m]$ and recurse for $A[i,m-1]$ and $A[m+1,j]$. If $P[m] \geq i$, we terminate. If we have access to the array P in time t, the query time complexity is $\mathrm{O}((1+k)t)$.

The algorithm is extended so that it works without storing P explicitly [26]. Instead we store a range minimum query data structure for P using $2n+\mathrm{o}(n)$ bits. The query algorithm is changed to work without accesses to P. Instead of checking if $P[m] < i$ for finding answers without duplicates, we use a bit array D whose length is the number of possible colors in A, and set $D[c] = 1$ if color c is output. Therefore it can be checked in constant time if a color is already output or not. After outputting all the answers, we clear the bits of D. To do this, we need to keep the output in the memory.

3 Enhancing Generalized Compressed Suffix Trees

3.1 New predecessor data structures

Let $T[1,n]$ be a string on alphabet $\mathcal{A}$ of size k. We give a simpler and more space-efficient index than [2] by omitting support for the access operation, which can be done using a GST. Our new index is summarized as follows.

▶ **Lemma 6.** *Given support for* select *in $f(n,k)$ time on a string S of length n on an alphabet of size k, there is a succinct index using $\mathrm{O}(n+k)$ bits that supports* rank*, predecessor, and successor for any character in $\mathrm{O}(f(n,k)\log\log k)$ time.*

Proof. Since predecessors and successors can be computed using rank and select operations as shown in Section 2.2, it is enough to show that we can compute ranks in $\mathrm{O}(f(n,k)\log\log k)$ time. We use a similar approach to [2]. We partition T into blocks $T_1, T_2, \ldots, T_m$ ($m = \lceil n/k \rceil$) of length k each. Let F_c be a bit-vector storing frequencies of $c \in \mathcal{A}$ for each block using unary codes. That is, $F_c = 1^{f_c(1)}01^{f_c(2)}0\cdots1^{f_c(m)}0$ where $f_c(i)$ is the number of occurrences of c in

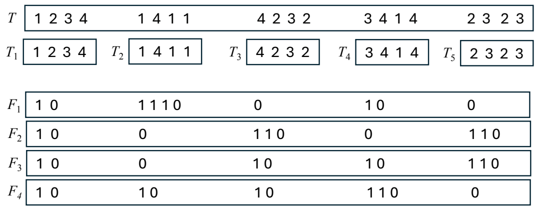


Figure 2 An example of our rank/select indexes of Lemma 6. The array T in the figure is the same as the array A in Figure 1.

T_i. The total length of F_c for all $c \in \mathcal{A}$ is $\sum_{c=1}^{k}(f_c+m) \le 2n+2k$ where $f_c = \sum_{i=1}^{m} f_c(i)$. We can compute the number of c's in $T_1, \dots, T_i$ in $f(n,k)$ time by $\mathrm{select}_0(F_c, i) - i$. We can also obtain the block containing the j-th occurrence of c in $f(n,k)$ time by $\mathrm{select}_1(F_c, j) - j + 1$. Therefore given an index j of T, we can obtain the number p of occurrences of c in blocks before the block containing j in $f(n,k)$ time by $p = \mathrm{select}_0(F_c, b) - b$ where $b = \lceil j/k \rceil$ is the index of the block. Then the problem is reduced to computing the rank of c in block T_b.

We assume that for each character $c \in \mathcal{A}$, we can compute select in τ time, and show that with this assumption we can compute rank in a block in $\mathrm{O}(\tau \log\log k)$ time using an auxiliary data structure of $\mathrm{O}(n)$ bits.

Let b be the index of the block containing $T[j]$ ($b = \lceil j/k \rceil$). If $f_c(b) \le \log k$, we can compute rank in $\mathrm{O}(\tau \log\log k)$ time by simply doing a binary search using select operations. If $f_c(b) > \log k$, we choose every $\log k$ values of the positions of c's in block T_b, and construct the y-fast trie [30]. The space is $\mathrm{O}\left(\left\lceil \frac{f_c(b)}{\log k} \right\rceil \cdot \log k\right)$ bits for each character c. Because there are less than $k/\log k$ such c's, the total space for block T_b is $\mathrm{O}\left(\sum_c \left(\frac{f_c(b)}{\log k} + 1\right) \log k\right) = \mathrm{O}(k)$ bits, and the total space for all the blocks is $\mathrm{O}(n)$ bits. Using the y-fast trie, we can obtain the predecessor among the samples in $\mathrm{O}(\log\log k)$ time, and then by a binary search we can obtain the true predecessor in $\mathrm{O}(\tau \log\log k)$ time. Successors and ranks are similarly computed in $\mathrm{O}(\tau \log\log k)$ time. ◀

3.2 Finding Closest Colored Suffixes

Given a suffix t of a string S_i and the index j of another string S_j, we compute a suffix of S_j that has the maximum LCP value with s.

We use the same framework as [18]. Let p and q be the lex-order of suffix s of S_i and the closest suffix t of S_j to t in the suffix array for $\mathcal{S}$, respectively. Then there are no suffixes of S_j whose lex-order in $\mathcal{S}$ is between p and q. Let $A[1, n+k]$ be an array of integers such that $A[i] = d$ if lexicographically the i-th suffix in $\mathcal{S}$ belongs to S_d. That is, $d = \mathrm{rank}_1(D, \mathrm{SA}_{\mathcal{S}}[i]) + 1$ is the ID of the string containing the suffix, as shown in Section 2.3. Then it holds that $A[q] = d$ and for all i between p and q exclusive $A[i] \ne d$. That is, q is either the predecessor or the successor of p representing d. In their paper, the data structure of [4] is used for computing predecessors and successors. We replace it with our predecessor data structure of Lemma 6. To use it, we need to give an algorithm for computing $\mathrm{select}_d(A, i)$.

▶ **Lemma 7.** *We can compute* $\mathrm{select}_d(A, i)$ *in* $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ *time.*

Proof. We can compute $\mathrm{select}_d(A, i)$ by the following formula.

$$\mathrm{select}_d(A, i) := \mathrm{ISA}_{\mathcal{S}}[\mathrm{SA}_{S_d}[i] + \mathrm{select}_1(D, d)]$$

This takes clearly $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. ◀

To compute the closest suffix we first compute the predecessor q_1 and the successor q_2 in A, which correspond to suffixes s_1 and s_2 of S_j, then we decide which one is closer to t. We compute the lengths of $\mathrm{LCP}(s_1, t)$ and $\mathrm{LCP}(s_2, t)$ and choose the larger one (if they are the same, we choose one arbitrarily). The length of $\mathrm{LCP}(s_1, t)$ is computed as follows.

- Find the leaves of the suffix tree of $\mathcal{S}$ corresponding to s_1 and t. They are lexicographically the q_1-th and p-th suffixes.
- Find the lowest common ancestor v between the leaves.
- Compute the string depth of v.

All this can be done in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ total time. The length of $\mathrm{LCP}(s_2, t)$ is computed similarly.

Now we give a proof of Theorem 1.

Proof. We construct compressed suffix trees for each of $S_1, S_2, \ldots, S_k$, and the compressed suffix tree for their concatenation $\mathcal{S}$. The compressed suffix arrays of S_i have size $\mathrm{SIZE}_{\mathrm{SA}}(|S_i|, \sigma)$ for $i = 1, 2, \ldots, k$ and the total size is $\mathrm{SIZE}_{\mathrm{SA}}(n, \sigma)$ bits. The compressed suffix array of $\mathcal{S}$ uses additional $\mathrm{SIZE}_{\mathrm{SA}}(n, \sigma)$ bits. The suffix tree structures are stored in $\mathrm{O}(n + k)$ bits. Therefore the total space is $\mathrm{SIZE}_{\mathrm{SA}}(2n, \sigma) + \mathrm{O}(n + k)$ bits. We store the bit-vector D of the lengths of the strings in $n + k + \mathrm{o}(n + k)$ bits. We also construct the predecessor data structure of Lemma 6 for the document array A storing ID's of suffixes in $\mathcal{S}$. Note that we do not store the document array A explicitly; each entry of A is computed in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time using the compressed suffix arrays of $\mathcal{S}$ (see Section 2.3).

For a query, we compute the global rank p of t in $\mathcal{S}$ using Equation 1 in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. Then we compute the predecessor q_1 and the successor q_2 in A such that $A[q_1] = A[q_2] = j$ in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log k)$ time using Lemma 6 where $f(n, k) = \mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$. We compute the LCP's between the suffix at q_1 and t and the suffix at q_2 and t in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time, and choose the suffix with longer LCP. The query complexity is $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log k)$ in total. ◀

3.3 Succinct index for weighted level ancestor queries

We prove Theorem 2. The solution of Kopelowitz and Lewenstein [15] for weighted level ancestor queries uses $\mathrm{O}(n \log n)$ bit space and supports a query in $\mathrm{O}(\log\log n)$ time. Though there are improved data structures [10, 3] that support a query in constant time for a suffix tree, they also use $\mathrm{O}(n \log n)$ bit space.

We give a succinct ($\mathrm{o}(n)$ bit) index for weighted level ancestors which can be used together with a (generalized) compressed suffix tree. The query time is $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \log\log n)$. The basic idea is to decompose the tree into small components using the tree covering [6] so that each component is a connected subgraph, called a mini tree, of the tree with $\mathrm{O}(\log^2 n)$ nodes. The number of components is $\mathrm{O}(n/\log^2 n)$. We create a tree, called tree over mini trees, connecting the components and use the $\mathrm{O}(n \log n)$ bit data structure (in our application we use $\mathrm{O}((n/\log^2 n) \log n) = \mathrm{O}(n/\log n)$ bits) for this new tree.

Given a query $\mathrm{WA}(v, w)$, we first find the mini tree containing v and check if the root of the mini tree has weight smaller than w. If so, the answer is inside the mini tree, and we can find it by a binary search using unweighted level ancestor queries. Because the mini tree contains $\mathrm{O}(\log^2 n)$ nodes, the path length from v to the mini tree root is also $\mathrm{O}(\log^2 n)$. Therefore the binary search takes $\mathrm{O}(\log\log n)$ steps and at each step we compute the string depth of a node in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. If the root of the mini tree has weight larger than w, we find the nearest mini tree whose root has weight smaller than w. This is done by using the data structure of [15] in $\mathrm{O}(\log\log n)$ time. Finally we find the answer - the right node of this mini tree through binary search. The total time complexity is $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log n)$ and the space complexity is $\mathrm{o}(n)$ bits in addition to that of the compressed suffix tree.

3.4 Document listing problem in a rooted tree

We generalize the document listing problem from arrays to rooted trees, that is, given two nodes of the tree with an ancestor-descendant relation we output all the distinct colors appearing on the path connecting them. A single node can have one color, multiple colors, or no color at all.

To solve this problem, we use the heavy-path decomposition of the rooted tree [27]. That is we decompose the tree with n nodes into heavy paths so that any root-to-leaf path intersects $\mathrm{O}(\log n)$ heavy paths.

First we give an $\mathrm{O}(n + k)$ bit representation of heavy paths. See Figure 3 for an example. We assume the tree topology is given as a BP sequence. Its length is at most $4(n+k)$ because the tree has $n + k$ leaves and at most $n + k - 1$ internal nodes. We encode heavy edges using bit-vector called "heavy". We set heavy$[i] = 1$ iff the edge between the node with preorder i and its parent is heavy. We mark a node if it is the head of a heavy path using a bit-vector of length n, called "head". Then by a nearest marked ancestor query we can find the preorder of the head of the heavy path containing a given node and the distance between them in constant time. We can give a total order for the heavy paths by the preorders of the head nodes. Using the bit-vector "head", we can give numbers from 1 to $m \leq n$ to heavy paths. We can also store the lengths of the heavy paths using unary codes in at most $m + n$ bits. This is encoded in "path-len". In Figure 3, the heavy path from node a to the 8-th leaf from the left has the head at node a, and it is the first heavy path because the head has the smallest preorder among all the five heavy paths. Its length 5 is encoded by the unary code at the beginning of the bit-vector "path-len".

Next we give an encoding of the colors of the nodes. For each heavy path, we encode the number of colors in each node using unary codes. The first heavy path has nodes a, c, e, f plus a leaf and the number of colors of them is 4, 2, 0, 1, and 0, respectively. The numbers are encoded in bit-vector "multi" using $n + u$ bits where $u \leq n$ is the total number of colors. For other heavy paths, the numbers of colors are stored similarly. The array named "color" stores the colors of nodes. Note that this array is constructed in the preprocessing phase and deleted after we construct the range minimum query data structure for array P (see Section 2.9). The range minimum query data structure uses $\mathrm{O}(n + u)$ bits. Note that in the original algorithm for arrays, we set $P[i] = -1$ if there are no values $j < i$ such that $A[j] = A[i]$, whereas in our algorithm for trees, we set $P[i] = j$ if there exists $j < i$ such that color$[j]$ = color$[i]$ and the node with color$[j]$ is the nearest such ancestor of the node with color$[i]$. In the example, for the heavy path containing nodes g, h, and a leaf, the colors of the nodes in the path are stored in color$[8, 9]$, and the corresponding P values are $3, 4$ because the root node has colors $3, 4$ and therefore we store the indices of P storing the same colors.

A document listing query is done as follows. We first give an algorithm for the case P is explicitly stored. We are given the head h and the tail t nodes of a sub-path p on a root-to-leaf path. First we partition the path into a set of heavy paths. This is done by first obtaining the nearest marked node of the tail node, that is, the head of the heavy path containing t. We go to the parent of the head node and repeat this process until we find the heavy path containing h. This is done in $\mathrm{O}(\log n)$ time because the sub-path p may contain $\mathrm{O}(\log n)$ heavy paths. Then for each heavy path which has an overlap with p, we find the minimum value $P[i]$ and choose the minimum among those values. We check whether the value of $P[i]$ is smaller than the index of h in the array color, and if it is, we output its color and continue the process. The first minimum value $P[i]$ is obtained in $\mathrm{O}(\log n)$ time, then we divide the heavy path containing $P[i]$ into two. To efficiently output all distinct

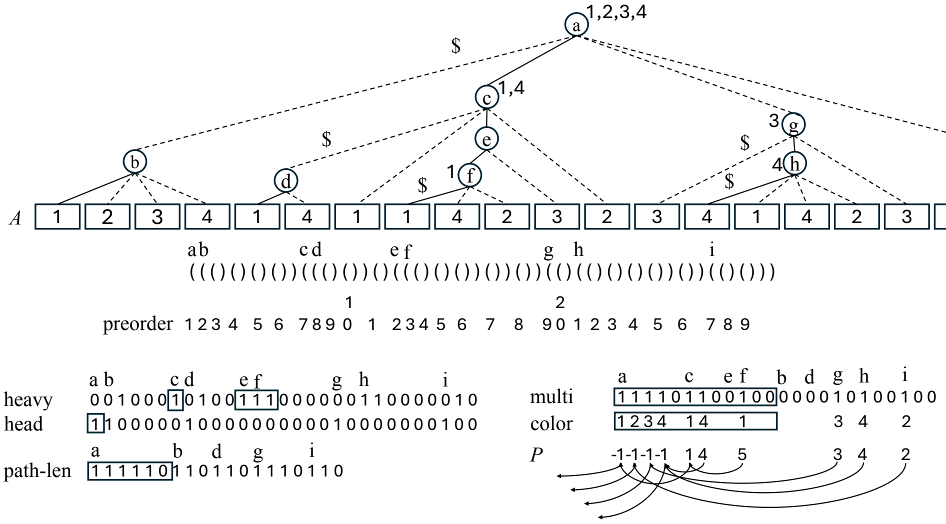


Figure 3 A data structure for document listing problem in a rooted tree. Node c has colors 1,4 because there are two suffixes of S_1 and S_4 ending at the node. The heavy-path from the node consists of nodes a, c, e, f, and a leaf, and it is represented by the boxes in the bit-vectors.

colors, we maintain divided paths using a Fibonacci heap [9]. Because $O(\log n + z)$ values are stored in the Fibonacci heap where z is the output size, the query algorithm runs in $O(\log n + z \log(z + \log n))$ time using $O(\log n(z + \log n))$ bit space.

We modify the algorithm for the case where P is represented only through its range minimum data structure. We compute the color of a suffix using a GST. In this case, we use a similar algorithm to the one described in Section 2.9 using a bit array of length k to mark output colors. First we obtain $O(\log n)$ paths representing p. Then for each path, we find the position i of the minimum value of P. We compute the color color$[i]$ in $O(\mathrm{TIME_{SA}})$ time using the GST. If this color was not found yet, we output it and divide the path into two. It is not necessary to find the minimum of the minimum values because duplication is checked by a different mechanism. It is also not necessary to store the paths in a Fibonacci heap. The time complexity is $O(\mathrm{TIME_{SA}}(\log n + z))$ and the working space is $O(\log n(\log n + z))$ bits.

4 Application to Suffix-prefix Matching

In the Suffix-Prefix problem we are given a set of strings $S_1, \dots, S_k$. We want to preprocess this set of strings so that given $i, j \in [1, k]$ we can answer query "what is the length of the longest suffix of S_i that is also a prefix of S_j" fast.

For any $i, j \in [1, k]$, we define $\mathrm{SPL}_{i,j}$ as the longest string that is both a suffix of S_i and a prefix of S_j. We consider the following variants [18]:

- One-to-One(i, j): output $\mathrm{SPL}_{i,j}$.
- One-to-All(i): output $\mathrm{SPL}_{i,j}$ for every $j \in [1, k]$.
- Report(i, ℓ): output all distinct $j \in [1, k]$ such that $\mathrm{SPL}_{i,j} \geq \ell$, where $\ell \geq 0$ is an integer.
- Count(i, ℓ): output the number of distinct $j \in [1, k]$ such that $\mathrm{SPL}_{i,j} \geq \ell$, where $\ell \geq 0$ is an integer.
- Top(i, K): output K distinct $j \in [1, k]$ with the highest values of $\mathrm{SPL}_{i,j}$, breaking ties arbitrarily.

Table 2 Time complexities for suffix-prefix problems. An existing solution uses $\mathrm{O}(n \log n)$ bits of space, while ours uses $\mathrm{SIZE}_{\mathrm{SA}}(2n, \sigma) + \mathrm{O}(n + k)$ bits of space. Typical values of $\mathrm{SIZE}_{\mathrm{SA}}$ and $\mathrm{TIME}_{\mathrm{SA}}$ are $n \log \sigma + \mathrm{O}(n)$ and $\mathrm{O}(\log n)$, respectively. The term z in the time complexity of the Report and Count queries is the output size of the Report query.

Query	Time ([18])	Time (ours)
One-to-One(i, j)	$\mathrm{O}(\log \log k)$	$\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$
One-to-All(i)	$\mathrm{O}(k)$	$\mathrm{O}(k \cdot \mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$
Report(i, ℓ)	$\mathrm{O}(\log n / \log \log n + z)$	$\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}}(\log n + z))$
Count(i, ℓ)	$\mathrm{O}(\log n / \log \log n)$	$\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}}(\log n + z))$

We give compact data structures for these problems except for Top(i, K). The results are summarized in Table 2.

4.1 Answering One-to-One and One-to-All queries

The base of the data structure consists of suffix trees ST_i of S_i for each $i \in [1, k]$ and a generalized suffix tree ST of the whole set of strings. ST is additionally enhanced with a rank-select queries data structure and the lowest common ancestor queries data structure. A node v of ST is colored j if the string label of v is equal to a suffix of j. Note that a node may have multiple colors.

Using Theorem 1 for the full string S_i and j we obtain the location of the closest suffix U of S_j in ST in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log k)$ time. We can convert the global rank of U to the local rank in ST_j in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. Next, using the lowest common ancestor query for S_i and U in ST we can find the LCP of those two strings, that is the string depth of the lowest common ancestor of the nodes representing them, in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. Next by using the weighted level ancestor query in ST_j for the leaf representing U and the LCP length we locate the node u of ST_j with the property that every ancestor of S_i in ST marked with color j is also an ancestor of u in ST_j, and every ancestor of u in ST_j is also an ancestor of S_i, in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$ time. Thus we reduced the problem of finding the nearest ancestor marked with color j in ST to finding the nearest marked ancestor in ST_j - that is in a situation where all the marks have the same color.

For the topology of each ST_j, we use the nearest marked ancestor data structure [28]. The additional space is $n + \mathrm{o}(n)$ bits for all ST_j's, and the answer is obtained in constant time. In summary, a One-to-One suffix-prefix query is done in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$ time.

For a One-to-All query, we naively repeat One-to-One queries for each $j \in [1, k]$. Then the time complexity is $\mathrm{O}(k \cdot \mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$.

4.2 Report and Count queries

As shown in [18], Report(i, ℓ) and Count(i, ℓ) are the same as Report$^r(j, \ell)$ and Count$^r(j, \ell)$ for the reversed strings $S_1^r, \ldots, S_k^r$. Let ST^r be the generalized compressed suffix tree of the set of the reversed strings, and let v be the nearest node to the root that is on the path from the root node representing S_j^r (reversed sting S_j) and that has a string depth at least ℓ. We color node u by color i if a suffix of S_i^r (without the terminator) ends at u. Then Report$^r(j, \ell)$ is to output all distinct colors on the path from v to the leaf corresponding to S_j^r. That is, Report$^r(j, \ell)$ corresponds to the document listing problem in a tree. The node v is obtained in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log \log n)$ time using the weighted level ancestor query. We use the algorithm from Section 3.4. Note that the arrays "color" and P are not stored explicitly. By using range minimum queries on P, we obtain only the position in P of the minimum value. To obtain the color, we use the compressed suffix tree. If a node u has color i, then u has an edge labeled \$ and a leaf connected by the edge stores a suffix of S_i^r (nodes f, g, h, and i in Figure 3). If u has multiple colors, we create a child w of u connected by an edge labeled \$ and create a leaf as a child of w for each color (nodes a and c in Figure 3). Since we can obtain the global rank of the suffix using the BP sequence of the generalized suffix tree, we can obtain the color in $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}})$ time. The total time complexity becomes $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}}(\log n + z))$ time. $\mathrm{Count}(j, \ell)$ is done in the same time complexity as $\mathrm{Report}(j, \ell)$.

For Top-K, we can use the observation in [18] that there exists an integer $\ell \in [0, n-1]$ such that $\mathrm{Count}(i, \ell+1) \leq K < \mathrm{Count}(i, \ell)$. Therefore we can solve a Top-K query by a binary search based on the value of $\mathrm{Count}(i, \ell)$. Unfortunately the time for $\mathrm{Count}(j, \ell)$ by our algorithm depends on the value, hence such an algorithm for Top-K is inefficient.

5 Concluding Remarks

This paper has proposed auxiliary data structures to enhance generalized compressed suffix trees (GSTs). By adding $\mathrm{O}(n)$ bits of space, we improved the time complexity for finding the closest colored suffix from $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log n)$ to $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log k)$ time, where k is the number of strings and n is the total length of the strings, and $\mathrm{TIME}_{\mathrm{SA}}$ is the time to obtain an entry of the suffix array. We also improved the time complexity for finding weighted level ancestors in a compressed suffix tree from $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log n)$ to $\mathrm{O}(\mathrm{TIME}_{\mathrm{SA}} \cdot \log\log n)$ time. Using these enhanced GSTs, we obtained linear space ($\mathrm{O}(n \log \sigma)$ bits) data structures for suffix-prefix queries for a set of strings. Future work will be to give time-efficient algorithms for Count and Top-K queries using $\mathrm{O}(n)$ bits of space. A challenging open problem is to obtain a weighted level ancestor data structure for suffix trees using $\mathrm{O}(n)$ bits of space supporting a query in constant time.

References

1 Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004. The 9th International Symposium on String Processing and Information Retrieval. doi:10.1016/S1570-8667(03)00065-0.

2 Jérémy Barbay, Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct indexes for strings, binary relations and multilabeled trees. *ACM Trans. Algorithms*, 7(4), September 2011. doi:10.1145/2000807.2000820.

3 Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. Weighted Ancestors in Suffix Trees Revisited. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2021.8.

4 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4), April 2015. doi:10.1145/2629339.

5 Arthur L. Delcher, Adam Phillippy, Jane Carlton, and Steven L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, June 2002. doi:10.1093/nar/30.11.2478.

6 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014. doi:10.1007/S00453-012-9664-0.

7 P. Ferragina and G. Manzini. Indexing compressed texts. *Journal of the ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

8 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 459–470, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74450-4_41.

9 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987. doi:10.1145/28869.28874.

10 Paweł Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 455–466, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

11 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 368–373, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109599.

12 R. Grossi, A. Gupta, and J. S. Vitter. High-Order Entropy-Compressed Text Indexes. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, 2003.

13 R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM Journal on Computing*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

14 D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

15 Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 565–574, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283444.

16 S. Kurtz. Reducing the Space Requirement of Suffix Trees. *Software – Practice and Experience*, 29(13):1149–1171, 1999. doi:10.1002/(SICI)1097-024X(199911)29:13\%3C1149::AID-SPE274\%3E3.0.CO;2-O.

17 Stefan Kurtz, Adam Phillippy, Arthur Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5:R12, February 2004. doi:10.1186/gb-2004-5-2-r12.

18 Grigorios Loukides, Solon P. Pissis, Sharma V. Thankachan, and Wiktor Zuba. Suffix-Prefix Queries on a Dictionary. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023)*, volume 259 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2023.21.

19 U. Manber and G. Myers. Suffix arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

20 J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

21 J. I. Munro, V. Raman, and S. R. Satti. Space Efficient Suffix Trees. *Journal of Algorithms*, 39:205–222, 2001. doi:10.1006/JAGM.2000.1151.

22 S. Muthukrishnan. Efficient Algorithms for Document Retrieval Problems. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545469.

23 G. Navarro and K. Sadakane. Fully-Functional Static and Dynamic Succinct Trees. *ACM Transactions on Algorithms (TALG)*, 10(3):Article No. 16, 39 pages, 2014.

24 R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4), 2007. doi:10.1145/1290672.1290680.

25 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/S00224-006-1198-X.

26 Kunihiko Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discrete Algorithms*, 5(1):12–22, 2007. doi:10.1016/J.JDA.2006.03.011.

27 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

28 Dekel Tsur. Succinct data structures for nearest colored node in a tree. *Information Processing Letters*, 132:6–10, 2018. doi:10.1016/j.ipl.2017.10.001.

29 P. Weiner. Linear Pattern Matching Algorithms. In *Proceedings of IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

30 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

31 Wiktor Zuba, Grigorios Loukides, Solon P. Pissis, and Sharma V. Thankachan. Approximate suffix-prefix dictionary queries. In Rastislav Královic and Antonín Kucera, editors, *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024, August 26-30, 2024, Bratislava, Slovakia*, volume 306 of *LIPIcs*, pages 85:1–85:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.85.

Tight (Double) Exponential Bounds for Identification Problems: Locating-Dominating Set and Test Cover

Dipayan Chakraborty
Université Clermont Auvergne, CNRS, Mines Saint-Étienne, Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France
Department of Mathematics and Applied Mathematics, University of Johannesburg, South Africa

Florent Foucaud
Université Clermont Auvergne, CNRS, Mines Saint-Étienne, Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France

Diptapriyo Majumdar
Indraprastha Institute of Information Technology Delhi, New Delhi, India

Prafullkumar Tale
Indian Institute of Science Education and Research Bhopal, India

Abstract

Foucaud et al. [ICALP 2024] demonstrated that some problems in NP can admit (tight) double-exponential lower bounds when parameterized by treewidth or vertex cover number. They showed these first-of-their-kind results by proving conditional lower bounds for certain graph problems, in particular, the metric-based identification problems (Strong) Metric Dimension. We continue this line of research and highlight the usefulness of this type of problems, to prove relatively rare types of (tight) lower bounds. We investigate fine-grained algorithmic aspects of classical (non-metric based) identification problems in graphs, namely Locating-Dominating Set, and in set systems, namely Test Cover. In the first problem, an input is a graph G on n vertices and an integer k, and the objective is to decide whether there is a subset S of k vertices such that any two distinct vertices not in S are dominated by distinct subsets of S. In the second problem, an input is a set of items U, a collection of subsets $\mathcal{F}$ of U called *tests*, and an integer k, and the objective is to select a set S of at most k tests such that any two distinct items are contained in a distinct subset of tests of S.

For our first result, we adapt the techniques introduced by Foucaud et al. [ICALP 2024] to prove similar (tight) lower bounds for these two problems.

- Locating-Dominating Set (respectively, Test Cover) parameterized by the treewidth of the input graph (respectively, the natural auxiliary graph) does not admit an algorithm running in time $2^{2^{o(\text{tw})}} \cdot \text{poly}(n)$ (respectively, $2^{2^{o(\text{tw})}} \cdot \text{poly}(|U| + |\mathcal{F}|)$), unless the ETH fails.

This augments the short list of NP-Complete problems that admit tight double-exponential lower bounds when parameterized by treewidth, and shows that "local" (non-metric-based) problems can also admit such bounds. We show that these lower bounds are tight by designing treewidth-based dynamic programming schemes with matching running times.

Next, we prove that these two problems also admit "exotic" (and tight) lower bounds, when parameterized by the solution size k. We prove that unless the ETH fails,

- Locating-Dominating Set does not admit an algorithm running in time $2^{o(k^2)} \cdot \text{poly}(n)$, nor a polynomial-time kernelization algorithm that reduces the solution size and outputs a kernel with $2^{o(k)}$ vertices, and
- Test Cover does not admit an algorithm running in time $2^{2^{o(k)}} \cdot \text{poly}(|U| + |\mathcal{F}|)$ nor a kernel with $2^{2^{o(k)}}$ vertices.

Again, we show that these lower bounds are tight by designing (kernelization) algorithms with matching running times. To the best of our knowledge, Locating-Dominating Set is the first known problem which is FPT when parameterized by solution size k, where the optimal running time has a quadratic function in the exponent. These results also extend the (very) small list of problems that admit an ETH-based lower bound on the number of vertices in a kernel, and (for

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 19; pp. 19:1–19:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Test Cover) a double-exponential lower bound when parameterized by the solution size. Whereas it is the first example, to the best of our knowledge, that admit a double exponential lower bound for the number of vertices.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Identification Problems, Locating-Dominating Set, Test Cover, Double Exponential Lower Bound, ETH, Kernelization Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.19

Related Version *Full Version*: https://arxiv.org/abs/2402.08346

Funding *Dipayan Chakraborty*: International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25.
Florent Foucaud: ANR project GRALMECO (ANR-21-CE48-0004), French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25), International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25.
Diptapriyo Majumdar: Supported by Science and Engineering Research Board (SERB) grant SRG/2023/001592.
Prafullkumar Tale: Supported by INSPIRE Faculty Fellowship DST/INSPIRE/04/2021/00314.

1 Introduction

The article aims to study the algorithmic properties of certain identification problems in discrete structures. In identification problems, one wishes to select a solution substructure of an input structure (a subset of vertices, the coloring of a graph, etc.) so that the solution substructure uniquely identifies each element. Some well-studied examples are, for example, the problems Test Cover for set systems and Metric Dimension for graphs (Problems [SP6] and [GT61] in the book by Garey and Johnson [39], respectively). This type of problem has been studied since the 1960s both in the combinatorics community (see e.g. Rényi [60] or Bondy [9]), and in the algorithms community since the 1970s [7, 10, 25, 55]. They have multiple practical and theoretical applications, such as network monitoring [59], medical diagnosis [55], bioinformatics [7], coin-weighing problems [62], graph isomorphism [4], games [19], machine learning [18] etc. An online bibliography on the topic with over 500 entries as of 2024 is maintained at [46].

In this article, we investigate fine-grained algorithmic aspects of identification problems in graphs, namely Locating-Dominating Set, and in set systems, namely Test Cover. Like most other interesting and practically motivated computational problems, identification problems also turned out to be NP-hard, even in very restricted settings. See, for example, [20] and [39], respectively. We refer the reader to "Related Work" towards the end of this section for a more detailed overview on their algorithmic complexity.

To cope with this hardness, these problems have been studied through the lens of parameterized complexity. In this paradigm, we associate each instance I with a parameter ℓ, and are interested to know whether the problem admits a *fixed parameter tractable* (FPT) algorithm, i.e., an algorithm with the running time $f(\ell) \cdot |I|^{\mathcal{O}(1)}$, for some computable function f. A parameter can either originate from the formulation of the problem itself or can be a property of the input. If a parameter originates from the formulation of the problem itself, then that is called a *natural parameter*. Otherwise, the parameters that are properties of the input graph are called the *structural parameters*. One of the most well-studied structural parameters is 'treewidth' (which, informally, quantifies how close the input graph is to a tree, and is denoted by tw). We refer readers to [26, Chapter 7] for a formal definition. Courcelle's celebrated theorem [21] states that the class of graph problems expressible in Monadic Second-Order Logic (MSOL) of constant size admit an algorithm running in time $f(\mathsf{tw}) \cdot \mathsf{poly}(n)$. Hence, a large class of problems admit an FPT algorithm when parameterized by the treewidth. Unfortunately, the function f is a tower of exponents whose height depends roughly on the size of the MSOL formula. Hence, this result serves as a starting point to obtain an (usually impractical) FPT algorithm.

Over the years, researchers have searched for more efficient problem-specific algorithms when parameterized by the treewidth. There is a rich collection of problems that admit an FPT algorithm with single- or almost-single-exponential dependency with respect to treewidth, i.e., of the form $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ or $2^{\mathcal{O}(\mathsf{tw} \log(\mathsf{tw}))} \cdot n^{\mathcal{O}(1)}$, (see, for example, [26, Chapter 7]). There are a handful of graph problems that only admit FPT algorithms with double- or triple-exponential dependence in the treewidth [8, 30, 31, 32, 42, 54]. In the respective articles, the authors prove that this double- (respectively, triple-) dependence in the treewidth cannot be improved unless the Exponential Time Hypothesis (ETH)[1] fails.

All the double- (or triple-) exponential lower bounds in treewidth mentioned in the previous paragraph are for problems that are #NP-complete, Σ_2^p-complete, or Π_2^p-complete. Indeed, until recently, this type of lower bounds were known only for problems that are complete for levels that are higher than NP in the polynomial hierarchy. Foucaud et al. [36] recently proved for the first time, that it is not necessary to go to higher levels of the polynomial hierarchy to achieve double-exponential lower bounds in the treewidth. The authors studied three NP-complete *metric-based graph problems* viz Metric Dimension, Strong Metric Dimension, and Geodetic Set. They proved that these problems admit double-exponential lower bounds in tw (and, in fact in the size of minimum vertex cover size vc for the second problem) under the ETH. The first two of these three problems are identification problems.

In this article, we continue this line of research and highlight the usefulness of identification problems to prove relatively rare types of lower bounds, by investigating fine-grained algorithmic aspects of Locating-Dominating Set and Test Cover, two classical (non-metric-based) identification problems. This also shows that this type of bounds can hold for "local" (i.e., non-metric-based) problems (the problems studied in [36] were all metric-based). Apart from serving as examples for double-exponential dependence on treewidth, these problems are of interest in their own right, and possess a rich literature both in the algorithms and discrete mathematics communities, as highlighted in "Related Work".

Locating-Dominating Set
Input: A graph G on n vertices and an integer k.
Question: Does there exist a locating-dominating set of size k in G, that is, a set S of $V(G)$ of size at most k such that for any two different vertices $u, v \in V(G) \setminus S$, their neighbourhoods in S are different, i.e., $N(u) \cap S \neq N(v) \cap S$ and non-empty?

Test Cover
Input: A set of items U, a collection $\mathcal{F}$ of subsets of U called *tests*, and an integer k.
Question: Does there exist a collection of at most k tests such that for each pair of items, there is a test that contains exactly one of the two items?

[1] The ETH roughly states that n-variable 3-SAT cannot be solved in time $2^{o(n)}n^{\mathcal{O}(1)}$. See [26, Chapter 14].

As Test Cover is defined over set systems, for structural parameters, we define an *auxiliary graph* in the natural way: A bipartite graph G on n vertices with bipartition $\langle R, B\rangle$ of $V(G)$ such that sets R and B contain a vertex for every set in $\mathcal{F}$ and for every item in U, respectively, and $r \in R$ and $b \in B$ are adjacent if and only if the set corresponding to r contains the element corresponding to b.

The Locating-Dominating Set problem is also a *graph domination problem.* In the classical Dominating Set problem, an input is an undirected graph G and an integer k, and the objective is to decide whether there is a subset $S \subseteq V(G)$ of size k such that for any vertex $u \in V(G) \setminus S$, at least one of its neighbours is in S. It can also be seen as a local version of Metric Dimension[2] in which the input is the same and the objective is to determine a set S of $V(G)$ such that for any two vertices $u, v \in V(G) \setminus S$, there exists a vertex $s \in S$ such that $dist(u, s) \neq dist(v, s)$.

We demonstrate the applicability of the techniques from [36] to Locating-Dominating Set and Test Cover. We adopt the main technique developed in [36] to our setting, namely, the *bit-representation gadgets* and *set representation gadget* to prove the following result.

▶ **Theorem 1.** *Unless the ETH fails, Locating-Dominating Set (respectively, Test Cover) parameterized by the treewidth of the input graph (respectively, the natural auxiliary graph) does not admit an algorithm running in time* $2^{2^{o(\mathsf{tw})}} \cdot \mathrm{poly}(n)$.

We remark that the algorithmic lower bound of Theorem 1 holds true even with respect to treedepth (and hence with respect to pathwidth), a parameter larger than treewidth. In contrast, Dominating Set admits an algorithm running in time $\mathcal{O}(3^{\mathsf{tw}} \cdot n^2)$ [67, 52]. In the full version of the paper, we prove that both Locating-Dominating Set and Test Cover admit an algorithm with matching running time, by nontrivial dynamic programming schemes on tree decompositions.

Theorem 1 adds Locating-Dominating Set and Test Cover to the short list of NP-Complete problems that admit (tight) double-exponential lower bounds for treewidth. Using the techniques mentioned in [36], two more problems, viz. Non-Clashing Teaching Map and Non-Clashing Teaching Dimension, from learning theory were recently shown in [14] to admit similar lower bounds.

Next, we prove that Locating-Dominating Set and Test Cover also admit "exotic" lower bounds, when parameterized by the solution size k. First, note that both problems are trivially FPT when parameterized by the solution size. Indeed, as any solution must have size at least logarithmic in the number of elements/vertices (assuming no redundancy in the input), the whole instance is a trivial single-exponential kernel for Locating-Dominating Set, and double-exponential in the case of Test Cover. To see this, note that in both problems, any two vertices/items must be assigned a distinct subset from the solution set. Hence, if there are more than 2^k of them, we can safely reject the instance. Thus, for Locating-Dominating Set, we can assume that the graph has at most $2^k + k$ vertices, and for Test Cover, at most 2^k items. Moreover, for Test Cover, one can also assume that every test is unique (otherwise, delete any redundant test), in which case there are at most 2^{2^k} tests. Hence, Locating-Dominating Set admits a kernel with size $\mathcal{O}(2^k)$, and an FPT algorithm running in time $2^{\mathcal{O}(k^2)}$ (See Proposition 7). We prove that both of these bounds are optimal.

[2] Note that Metric Dimension is also an identification problem, but it is inherently non-local in nature, and indeed was studied together with two other non-local problems in [36], where the similarities between these non-local problems were noticed.

▶ **Theorem 2.** *Unless the ETH fails, Locating-Dominating Set, parameterized by the solution size k, does not admit*

- *an algorithm running in time $2^{o(k^2)} \cdot n^{\mathcal{O}(1)}$, nor*
- *a polynomial time kernelization algorithm that reduces the solution size and outputs a kernel with $2^{o(k)}$ vertices.*

To the best of our knowledge, Locating-Dominating Set is the first known problem to admit such an algorithmic lower bound, with a matching upper bound, when parameterized by the solution size. The only other problems known to us, admitting similar lower bounds, are for structural parameterizations like vertex cover [1, 14, 36] or pathwidth [58, 61]. The second result is also quite rare in the literature. The only results known to us about ETH-based conditional lower bounds on the number of vertices in a kernel when parameterized by the solution size are for Edge Clique Cover [27] and Biclique Cover [15][3]. Theorem 2 also improves upon a "no $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ algorithm" bound from [5] (under $\mathsf{W}[2] \neq \mathsf{FPT}$) and a $2^{o(k \log k)}$ ETH-based lower bound recently proved in [11].

Now, consider the case of Test Cover. As mentioned before, it is safe to assume that $|\mathcal{F}| \leq 2^{|U|}$ and $|U| \leq 2^k$. By Bondy's celebrated theorem [9], which asserts that in any feasible instance of Test Cover, there is always a solution of size at most $|U| - 1$, we can also assume that $k \leq |U| - 1$. Hence, the brute-force algorithm that enumerates all the sub-collections of tests of size at most k runs in time $|\mathcal{F}|^{\mathcal{O}(|U|)} = 2^{\mathcal{O}(|U|^2)} = 2^{2^{\mathcal{O}(k)}}$. Our next result proves that this simple algorithm is again optimal.

▶ **Theorem 3.** *Unless the ETH fails, Test Cover does not admit*

- *an algorithm running in time $2^{2^{o(k)}} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, nor*
- *a polynomial time kernelization algorithm that reduces the solution size and outputs a kernel with $2^{2^{o(k)}}$ vertices.*

This result adds Test Cover to the relatively rare list of NP-complete problems that admit such double-exponential lower bounds when parameterized by the solution size and the matching algorithm. The only other examples that we know of are Edge Clique Cover [27], Distinct Vectors Problem [57], and Telephone Broadcast [66]. For double-exponential algorithmic lower bounds with respect to structural parameters, please see [33, 42, 45, 47, 48, 50, 51, 53].

The second result in the theorem is a simple corollary of the first result. Assume that the problem admits a kernel with $2^{2^{o(k)}}$ vertices. Then, the brute-force enumerating all the possible solutions works in time $\binom{2^{2^{o(k)}}}{k} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, which is $2^{k \cdot 2^{o(k)}} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, which is $2^{2^{o(k)}} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, contradicting the first result. To the best of our knowledge, Test Cover is the first problem that admit a double exponential kernelization lower bound for the number of vertices when parameterized by solution size, or by any natural parameter.

Related Work. Locating-Dominating Set was introduced by Slater in the 1980s [63, 64]. The problem is NP-complete [20], even for special graph classes such as planar unit disk graphs [56], planar bipartite subcubic graphs, chordal bipartite graphs, split graphs and co-bipartite graphs [35], interval and permutation graphs of diameter 2 [38]. By a straightforward application of Courcelle's theorem [22], Locating-Dominating Set is FPT for parameter treewidth and even cliquewidth [23]. Explicit polynomial-time algorithms were given for trees [63], block graphs [3], series-parallel graphs [20], and cographs [37]. Regarding the approximation complexity of Locating-Dominating Set, see [35, 40, 65].

[3] Additionally, Point Line Cover does not admit a kernel with $\mathcal{O}(k^{2-\epsilon})$ *vertices*, for any $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP}/poly$ [49].

It was shown in [5] that Locating-Dominating Set cannot be solved in time $2^{o(n)}$ on bipartite graphs, nor in time $2^{o(\sqrt{n})}$ on planar bipartite graphs or on apex graphs, assuming the ETH. Moreover, they also showed that Locating-Dominating Set cannot be solved in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ on bipartite graphs, unless $\mathsf{W}[2] = \mathsf{FPT}$. Note that the authors of [5] have designed a complex framework with the goal of studying a large class of identification problems related to Locating-Dominating Set and similar problems.

In [12], structural parameterizations of Locating-Dominating Set were studied. It was shown that the problem admits a linear kernel for the parameter max-leaf number, however (under standard complexity assumptions) no polynomial kernel exists for the solution size, combined with either the vertex cover number or the distance to a clique. They also provide a double-exponential kernel for the parameter distance to the cluster. In the full version [11] of [12], the same authors show that Locating-Dominating Set does neither admit a $2^{o(k \log k)}n^{\mathcal{O}(1)}$-time nor an $n^{o(k)}$-time algorithm, assuming the ETH.

Test Cover was shown to be NP-complete by Garey and Johnson [39, Problem SP6] and it is also hard to approximate within a ratio of $(1 - \epsilon)\ln n$ [10] (an approximation algorithm with ratio $1 + \ln n$ exists by reduction to Set Cover [7]). As any solution has size at least $\log_2(n)$, the problem admits a trivial kernel of size 2^{2^k}, and thus Test Cover is FPT parameterized by solution size k. Test Cover was studied within the framework of "above/below guarantee" parameterizations in [6, 24, 25, 41] and kernelization in [6, 24, 41]. These results have shown an intriguing behavior for Test Cover, with some nontrivial techniques being developed to solve the problem [6, 25]. Test Cover is FPT for parameters $n-k$, but $W[1]$-hard for parameters $m-k$ and $k-\log_2(n)$ [25]. However, assuming standard assumptions, there is no polynomial kernel for the parameterizations by k and $n-k$ [41], although there exists a "partially polynomial kernel" for parameter $n-k$ [6] (i.e. one with $O((n-k)^7)$ elements, but potentially exponentially many tests). When the tests have all a fixed upper bound r on their size, the parameterizations by k, $n-k$ and $m-k$ all become FPT with a polynomial kernel [24, 41].

The problem Discriminating Code [16] is very similar to Test Cover (with the distinction that the input is presented as a bipartite graph, one part representing the elements and the other, the tests, and that every element has to be covered by some solution test), and has been shown to be NP-complete even for planar instances [17].

Organization. Due to the space constraints, we present overviews of the reductions in this extended abstract. Formal proofs for the arguments can be found in the full version of the paper. We use the Locating-Dominating Set problem to demonstrate key technical concepts regarding our lower bounds and algorithms. We present an overview of the arguments about Locating-Dominating Set in Sections 3 and 4. The arguments regarding Test Cover follow the identical line, and an overview is presented in Section 5. We conclude with an open problem in Section 6.

2 Preliminaries

For a positive integer q, we denote the set $\{1, 2, \ldots, q\}$ by $[q]$. We use $\mathbb{N}$ to denote the collection of all non-negative integers.

Graph theory. We use standard graph-theoretic notation, and we refer the reader to [28] for any undefined notation. For an undirected graph G, sets $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively. We denote an edge with two endpoints u, v as uv. Unless otherwise specified, we use n to denote the number of vertices in the input graph G of the problem under consideration. Two vertices u, v in $V(G)$ are *adjacent* if there is an edge uv in G. The *open neighborhood* of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v. The *closed neighborhood* of a vertex v, denoted by $N_G[v]$, is the set $N_G(v) \cup \{v\}$. We say that a vertex u is a *pendant vertex* if $|N_G(v)| = 1$. We omit the subscript in the notation for neighborhood if the graph under consideration is clear. For a subset S of $V(G)$, we define $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. For a subset S of $V(G)$, we denote the graph obtained by deleting S from G by $G - S$. We denote the subgraph of G induced on the set S by $G[S]$.

Locating-Dominating Sets. A subset of vertices S in graph G is called its *dominating set* if $N[S] = V(G)$. A dominating set S is said to be a *locating-dominating set* if for any two different vertices $u, v \in V(G) \setminus S$, we have $N(u) \cap S \neq N(v) \cap S$. In this case, we say vertices u and v are *distinguished* by the set S. We say a vertex u is *located* by set S if for any vertex $v \in V(G) \setminus \{u\}$, u and v are distinguished by S (equivalently $N(u) \cap S \neq N(v) \cap S$). Note that, if S locates u, then any superset $S' \supset S$ also locates u. By extension, a set X is *located* by S if all vertices in X are located by S. We note the following simple observation (see also [13, Lemma 5]).

▶ **Observation 4.** *If S is a locating-dominating set of a graph G, then there exists a locating-dominating set S' of G such that $|S'| \leq |S|$ and that contains all vertices that are adjacent with a pendant vertices (i.e. vertices of degree 1) in G.*

Proof. Let u be a pendant vertex which is adjacent with a vertex v of G. We now look for a locating dominating set S' of G such that $|S'| \leq |S|$ and contains the vertex v. As S is a (locating) dominating set, we have $\{u, v\} \cap S \neq \emptyset$. If $v \in S$, then take $S' = S$. Therefore, let us assume that $u \in S$ and $v \notin S$. Define $S' = (S \cup \{v\}) \setminus \{u\}$. It is easy to see that S' is a dominating set. If S' is not a locating-dominating set, then there exists w, apart from u, in the neighbourhood of v such that both u and w are adjacent with *only* v in S'. As u is a pendant vertex and v its unique neighbour, w is not adjacent to u. Hence, w was not adjacent with any vertex in $S' \setminus \{v\} = S \setminus \{u\}$. This, however, contradicts the fact that S is a (locating) dominating set. Hence, S' is a locating-dominating set and $|S'| = |S|$. Thus, the result follows from repeating this argument for each vertex of G adjacent to a pendant vertex. ◀

Parameterized complexity. An instance of a parameterized problem Π consists of an input I, which is an input of the non-parameterized version of the problem, and an integer k, which is called the *parameter*. Formally, $\Pi \subseteq \Sigma^* \times \mathbb{N}$. A problem Π is said to be *fixed-parameter tractable*, or FPT, if given an instance (I, k) of Π, we can decide whether (I, k) is a YES-instance of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$. Here, $f : \mathbb{N} \mapsto \mathbb{N}$ is some computable function depending only on k. A parameterized problem Π is said to admit a *kernelization* if given an instance (I, k) of Π, there is an algorithm that runs in time polynomial in $|I| + k$ and constructs an instance (I', k') of Π such that (i) $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$, and (ii) $|I'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \mapsto \mathbb{N}$ depending only on k. If $g(\cdot)$ is a polynomial function, then Π is said to admit a *polynomial* kernelization. For a detailed introduction to parameterized complexity and related terminologies, we refer the reader to the recent books by Cygan et al. [26] and Fomin et al. [34].

3 Locating-Dominating Set Parameterized by Treewidth

We first present a bird's eye overview of the dynamic programming algorithm. Let $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ be a nice tree decomposition of G. For every node $t \in V(T)$, consider the subtree T_t of T rooted at t. Let G_t denote the subgraph of G that is induced by the vertices that are present in the bags of T_t. For every node $t \in T$, we define a subproblem (or DP-state) using a tuple $[t, (Y, W), (\mathcal{A}, \mathcal{D}), \mathcal{B}]$. Consider a partition $(Y, W, X_t \setminus (Y \cup W))$ of X_t. The first part denotes the vertices in the partial solution S. The second part denotes the vertices in X_t that are dominated (but need not be located) by the solution vertices in G_t but that are outside X_t. To extend this partial solution, we need to keep track of vertices that are adjacent to a unique subset in Y. For example, suppose there is vertex $u \in V(G_t) \setminus (S \cup X_t)$ such that $N_{G_t}(u) \cap S = A$ for some subset $A \subseteq Y$. Then u still needs to be located by S. It means that there should not be a vertex, say v, in $V(G) \setminus V(G_t)$ such that $N_{G_t}(v) \cap S' = A$, where S' is an extension of the partial solution S. Hence, we need to keep track of all such vertices by keeping track of the neighbourhood of all such vertices. We define $\mathcal{A}$, which is a subset of the power set of Y, to store all such sets that are the neighborhoods of vertices in $V(G_t) \setminus X_t$. Similarly, we define $\mathcal{D}$ to store all such sets with respect to vertices that are in X_t. Finally, we define $\mathcal{B}$ to store the pairs of vertices that need to be resolved by the extension of the partial solution. We formalise these ideas in the full version of the paper to prove the following theorem.

▶ **Theorem 5.** *Locating-Dominating Set, parameterized by the treewidth* tw *of the input graph admits an algorithm running in time* $2^{2^{\mathcal{O}(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$*.*

In the remainder of this section, we prove the lower bound mentioned in Theorem 1 by presenting a reduction from a variant of 3-SAT called $(3,3)$-SAT. In this variation, an input is a boolean satisfiability formula ψ in conjunctive normal form such that each clause contains *at most* 3 variables, and each variable appears at most 3 times. Using the ETH [43], the sparcification lemma [44], and a simple reduction from 3-SAT, we have the following result.

▶ **Proposition 6.** *$(3,3)$-SAT, with n variables and m clauses, does not admit an algorithm running in time $2^{o(m+n)}$, unless the* ETH *fails.*

We highlight that every variable appears positively and negatively at least once. Otherwise, if a variable appears only positively (respectively, only negatively) then we can assign it `True` (respectively, `False`) and safely reduce the instance by removing the clauses containing this variable. Hence, instead of the first, second, or third appearance of the variable, we use the first positive, first negative, second positive, or second negative appearance of the variable.

Reduction. The reduction takes as input an instance ψ of $(3,3)$-SAT with n variables and outputs an instance (G, k) of Locating-Dominating Set such that $\mathtt{tw}(G) = \mathcal{O}(\log(n))$. Suppose $X = \{x_1, \ldots, x_n\}$ is the collection of variables and $C = \{C_1, \ldots, C_m\}$ is the collection of clauses in ψ. Here, we consider $\langle x_1, \ldots, x_n\rangle$ and $\langle C_1, \ldots, C_m\rangle$ to be arbitrary but fixed orderings of variables and clauses in ψ. For a particular clause, the first order specifies the first, second, or third (if it exists) variable in the clause in a natural way. The second ordering specifies the first/second positive/negative appearance of variables in X in a natural way. The reduction constructs a graph G as follows.

- To construct a variable gadget for x_i, it starts with two claws $\{\alpha_i^0, \alpha_i^1, \alpha_i^2, \alpha_i^3\}$ and $\{\beta_i^0, \beta_i^1, \beta_i^2, \beta_i^3\}$ centered at α_i^0 and β_i^0, respectively. It then adds four vertices $x_i^1, \neg x_i^1, x_i^2, \neg x_i^2$, and the corresponding edges, as shown in Figure 1. Let A_i be the collection of these twelve vertices and we define $A = \cup_{i=1}^{n} A_i$. Define $X_i := \{x_i^1, \neg x_i^1, x_i^2, \neg x_i^2\}$.

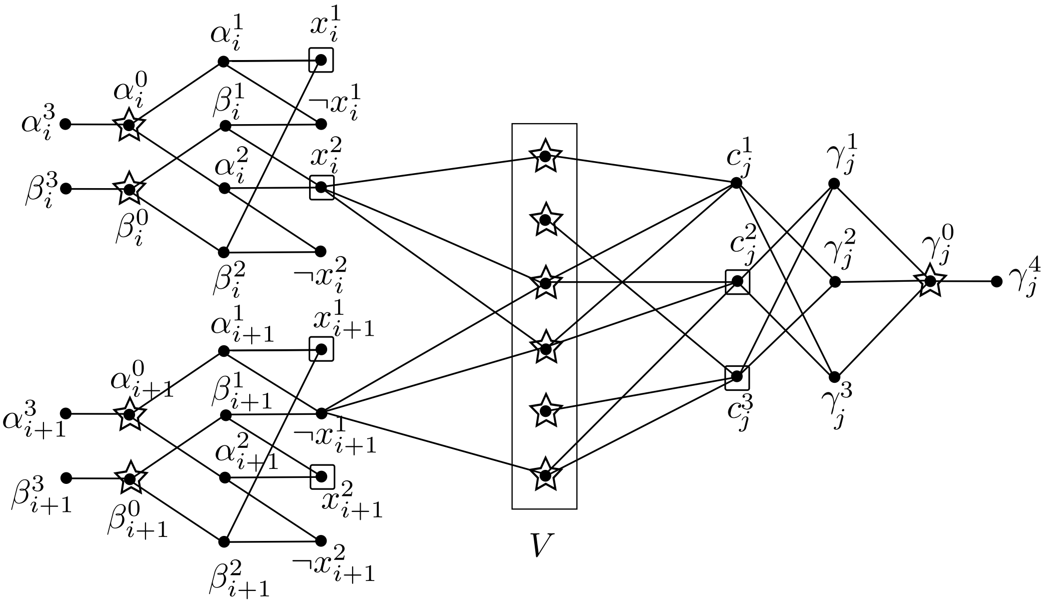


Figure 1 For the sake of clarity, we do not explicitly show the pendant vertices adjacent to vertices in V. The variable and clause gadgets are on the left-side and right-side of V, respectively. In this example, we consider a clause $C_j = x_i \vee \neg x_{i+1} \vee x_{i+2}$. Moreover, suppose this is the second positive appearance of x_i and the first negative appearance of x_{i+1}, and x_i corresponds to c_j^1 and x_{i+1} corresponds to c_j^2. Suppose V contains 6 vertices indexed from top to bottom, and the set corresponding to these two appearances are $\{1,3,4\}$ and $\{3,4,6\}$ respectively. The star boundary denote the vertices that we can assume to be in any locating-dominating set, without loss of generality. The square boundary corresponds to selection of other vertices in S. In the above example, it corresponds to setting both x_i and x_{i+1} to `True`. On the clause side, the selection corresponds to selecting x_i to satisfy the clause C_j.

- To construct a clause gadget for C_j, the reduction starts with a star graph centered at γ_j^0 and with four leaves $\{\gamma_j^1, \gamma_j^2, \gamma_j^3, \gamma_j^4\}$. It then adds three vertices c_j^1, c_j^2, c_j^3 and the corresponding edges shown in Figure 1. Let B_j be the collection of these eight vertices and $B = \cup_{j=1}^{m} B_j$.
- Let p be the smallest positive integer such that $4n \leq \binom{2p}{p}$. Define $\mathcal{F}_p$ as the collection of subsets of $[2p]$ that contains exactly p integers (such a collection $\mathcal{F}_p$ is called a *Sperner family*). Define $\texttt{set-rep} : \bigcup_{i=1}^{n} X_i \to \mathcal{F}_p$ as an injective function by arbitrarily assigning a set in $\mathcal{F}_p$ to a vertex x_i^ℓ or $\neg x_i^\ell$, for every $i \in [n]$ and $\ell \in [2]$. In other words, every appearance of a literal is assigned a distinct subset in $\mathcal{F}_p$.
- The reduction adds a *connection portal* V, which is a clique on $2p$ vertices $v_1, v_2, \ldots, v_{2p}$. For every vertex v_q in V, the reduction adds a pendant vertex u_q adjacent to v_q.
- For each vertex $x_i^\ell \in X$ where $i \in [n]$ and $\ell \in [2]$, the reduction adds edges (x_i^ℓ, v_q) for every $q \in \texttt{set-rep}(x_i^\ell)$. Similarly, it adds edges $(\neg x_i^\ell, v_q)$ for every $q \in \texttt{set-rep}(\neg x_i^\ell)$.
- For a clause C_j, suppose variable x_i appears positively for the ℓ^{th} time as the r^{th} variable in C_j. For example, x_i appears positively for the second time as the third variable in C_j. Then, the reduction adds edges across B and V such that the vertices c_j^r and x_i^ℓ have the same neighbourhood in V, namely, the set $\{v_q : q \in \texttt{set-rep}(x_i^\ell)\}$. Similarly, it adds edges for the negative appearance of the variables.

This concludes the construction of G. The reduction sets $k = 4n + 3m + 2p$ and returns (G, k) as the reduced instance of Locating-Dominating Set.

We now provide an overview of the proof of correctness in the reverse direction. The crux of the correctness is: Without loss of generality, all the vertices in the connection portal V are present in any locating-dominating set S of G. Consider a vertex, say x_i^1, on the "variable-side" of S and a vertex, say c_j^1, on the "clause-side" of S. If both of these vertices have the same neighbors in the connection portal and are not adjacent to the vertices in $S \setminus V$, then at least one of x_i^1 or c_j^1 must be included in S.

More formally, suppose S is a locating-dominating set of G of size at most $k = 4n + 3m + 2p$. Then, we prove that S must have exactly 4 vertices from each variable gadget and exactly 3 vertices from each clause gadget. Further, S contains either $\{\alpha_0^i, \beta_0^i, x_i^1, x_i^2\}$ or $\{\alpha_0^i, \beta_0^i, \neg x_i^1, \neg x_i^2\}$, but no other combination of vertices in the variable gadget corresponding to x_i. For a clause gadget corresponding to C_j, S contains either $\{\gamma_j^0, c_j^2, c_j^3\}$, $\{\gamma_j^0, c_j^1, c_j^3\}$, or $\{\gamma_j^0, c_j^1, c_j^2\}$, but no other combination. These choices imply that c_j^1, c_j^2, or c_j^3 are not adjacent to any vertex in $S \setminus V$. Consider the first case and suppose c_j^1 corresponds to the second positive appearance of variable x_i. By the construction, the neighborhoods of x_i^2 and c_j^1 in V are identical. This forces a selection of $\{\alpha_0^i, \beta_0^i, x_i^1, x_i^2\}$ in S from the variable gadget corresponding to x_i, which corresponding to setting x_i to `True`. Hence, a locating dominating set S of size at most k implies a satisfying assignment of ψ.

Sketch of Proof of Theorem 1. Note that since each component of $G - V$ is of constant order, the tree-width of G is $\mathcal{O}(|V|)$. By the asymptotic estimation of the central binomial coefficient, $\binom{2p}{p} \sim \frac{4^p}{\sqrt{\pi \cdot p}}$ [2]. To get the upper bound of $2p$, we scale down the asymptotic function and have $4n \leq \frac{4^p}{2^p} = 2^p$. Since we choose the value of p as small as possible such that $2^p \geq 4n$, we choose $p = \log(n) + 3$. This ensures us that $p = \mathcal{O}(\log(n))$. And hence, $|V| = \mathcal{O}(\log(n))$ which implies $\mathsf{tw}(G) = \mathcal{O}(\log n)$. The remaining arguments are standard for proving the conditional lower bound under ETH. ◀

4 Locating-Dominating Set Parameterized by the Solution Size

In this section, we study the parameterized complexity of Locating-Dominating Set when parameterized by the solution size k. First, we formally prove that the problem admits a kernel with $\mathcal{O}(2^k)$ vertices, and hence a simple FPT algorithm running in time $2^{\mathcal{O}(k^2)}$. Next, we prove that both results mentioned above are optimal under the ETH.

▶ **Proposition 7.** *Locating-Dominating Set admits a kernel with $\mathcal{O}(2^k)$ vertices and an algorithm running in time $2^{\mathcal{O}(k^2)} + \mathcal{O}(k \log n)$.*

Proof. Slater proved that for any graph G on n vertices with a locating-dominating set of size k, we have $n \leq 2^k + k - 1$ [64]. Hence, if $n > 2^k + k - 1$, we can return a trivial No instance (this check takes time $\mathcal{O}(k \log n)$). Otherwise, we have a kernel with $\mathcal{O}(2^k)$ vertices. In this case, we can enumerate all subsets of vertices of size k, and for each of them, check in quadratic time if it is a valid solution. Overall, this takes time $\binom{n}{k} n^2$; since $n \leq 2^k + k - 1$, this is $\binom{2^{\mathcal{O}(k)}}{k} \cdot 2^{\mathcal{O}(k)}$, which is $2^{\mathcal{O}(k^2)}$. ◀

To prove Theorem 2, we present a reduction that takes as input an instance ψ, with n variables, of 3-SAT and returns an instance (G, k) of Locating-Dominating Set such that $|V(G)| = 2^{\mathcal{O}(\sqrt{n})}$ and $k = \mathcal{O}(\sqrt{n})$. By adding dummy variables in each set, we can assume that $\sqrt{n}$ is an even integer. Suppose the variables are named $x_{i,j}$ for $i, j \in [\sqrt{n}]$. The reduction constructs graph G as follows:

- It partitions the variables of ψ into $\sqrt{n}$ many *buckets* $X_1, X_2, \ldots, X_{\sqrt{n}}$ such that each bucket contains exactly $\sqrt{n}$ many variables. Let $X_i = \{x_{i,j} \mid j \in [\sqrt{n}]\}$ for all $i \in [\sqrt{n}]$.

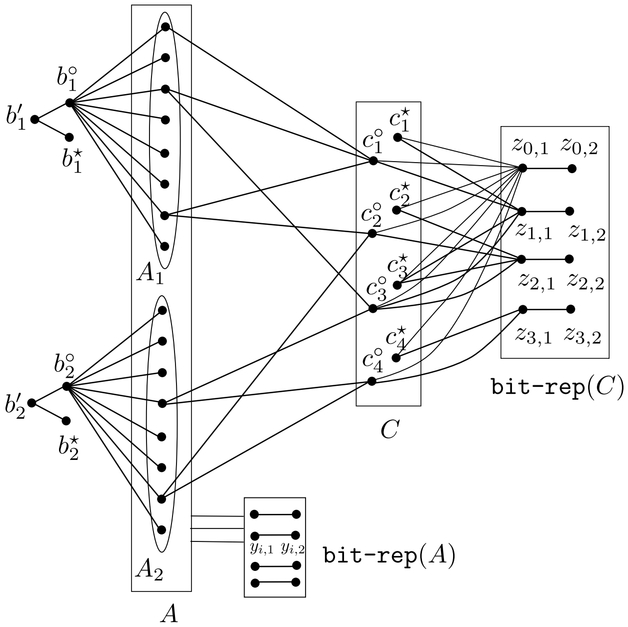


Figure 2 Suppose an instance ψ of 3-SAT has $n = 9$ variables and 4 clauses. We do not show the third variable bucket and explicit edges across A and `bit-rep`(A) for brevity.

  - For every X_i, it constructs set A_i of $2^{\sqrt{n}}$ new vertices, $A_i = \{a_{i,\ell} \mid \ell \in [2^{\sqrt{n}}]\}$. Each vertex in A_i corresponds to a unique assignment of variables in X_i. Let A be the collection of all the vertices added in this step.
  - For every X_i, the reduction adds a path on three vertices b_i°, b_i', and $b_i^\star$ with edges (b_i°, b_i') and $(b_i', b_i^\star)$. Suppose B is the collection of all the vertices added in this step.
  - For every X_i, it makes b_i° adjacent with every vertex in A_i.
- For every clause C_j, the reduction adds a pair of vertices $c_j^\circ, c_j^\star$. For a vertex $a_{i,\ell} \in A_i$ for some $i \in [\sqrt{n}]$, and $\ell \in [2^{\sqrt{n}}]$, if the assignment corresponding to vertex $a_{i,\ell}$ satisfies clause C_j, then it adds edge $(a_{i,\ell}, c_j^\circ)$.
- The reduction adds a bit-representation gadget[4] to locate set A. Once again, informally speaking, it adds some supplementary vertices such that it is safe to assume these vertices are present in a locating-dominating set, and they locate every vertex in A. More precisely:
  - First, set $q := \lceil \log(|A|) \rceil + 1$. This value for q allows to uniquely represent each integer in $[|A|]$ by its bit-representation in binary (starting from 1 and not 0).
  - For every $i \in [q]$, the reduction adds two vertices $y_{i,1}$ and $y_{i,2}$ and edge $(y_{i,1}, y_{i,2})$.
  - For every integer $q' \in [|A|]$, let `bit`(q') denote the binary representation of q' using q bits. Connect $a_{i,\ell} \in A$ with $y_{i,1}$ if the i^{th} bit in `bit`($(i + (\ell - 1) \cdot \sqrt{n})$) is 1.
  - It adds two vertices $y_{0,1}$ and $y_{0,2}$, and edge $(y_{0,1}, y_{0,2})$. It also makes every vertex in A adjacent with $y_{0,1}$.
    Let `bit-rep`(A) be the collection of the vertices $y_{i,1}$ for all $i \in \{0\} \cup [q]$ added in this step.
- Finally, the reduction adds a bit-representation gadget to locate set C. However, it adds the vertices in such a way that for any pair $c_j^\circ, c_j^\star$, the supplementary vertices adjacent to them are identical.

[4] With the problem-specific adaptations, the bit-representation gadgets resembles the gadget used in [29].

- The reduction sets $p := \lceil \log(|C|/2) \rceil + 1$ and for every $i \in [p]$, it adds two vertices $z_{i,1}$ and $z_{i,2}$ and edge $(z_{i,1}, z_{i,2})$.
- For every integer $j \in [|C|/2]$, let $\texttt{bit}(j)$ denote the binary representation of j using q bits. Connect $c^{\circ}_j, c^{\star}_j \in C$ with $z_{i,1}$ if the i^{th} bit in $\texttt{bit}(j)$ is 1.
- It adds two vertices $z_{0,1}$ and $z_{0,2}$, and edge $(z_{0,1}, z_{0,2})$. It also makes every vertex in C adjacent with $y_{0,1}$.

  Let $\texttt{bit-rep}(C)$ be the collection of the vertices $z_{i,1}$ for all $i \in \{0\} \cup [p]$ added in this step.

This completes the reduction. See Figure 2 for an illustration. Please check this. The reduction sets

$$k = |B|/3 + (\lceil \log(|A|) \rceil + 1 + 1) + \lceil (\log(|C|/2) \rceil + 1 + 1) + \sqrt{n} = \mathcal{O}(\sqrt{n})$$

as $|B| = 3\sqrt{n}$, $|A| = \sqrt{n} \cdot 2^{\sqrt{n}}$, and $|C| = \mathcal{O}(n^3)$, and returns (G, k) as a reduced instance.

We present a brief overview of the proof of correctness in the reverse direction. Suppose S is a locating-dominating set of graph G of size at most k. Note that $b^{\star}_i$, $y_{i,2}$ and $z_{i,2}$ are pendant vertices for appropriate i. We argue that it is safe to consider that vertices b'_i, $y_{i,1}$, and $z_{i,1}$ are in S. This already forces $|B|/2 + \lceil \log(|A|) \rceil + 2 + \lceil \log(|C|/2) \rceil + 2$ many vertices in S. The remaining $\sqrt{n}$ many vertices need to locate vertices in pairs $(b^{\circ}_i, b^{\star}_i)$ and $(c^{\circ}_j, c^{\star}_j)$ for every $i \in [\sqrt{n}]$ and $j \in [|C|]$. Note that the only vertices that are adjacent with b°_i but are *not* adjacent with $b^{\star}_i$ are in A_i. Also, the only vertices that are adjacent with c°_j but are *not* adjacent with $c^{\star}_j$ are in A_i and correspond to an assignment that satisfies C_j. Hence, any locating-dominating set should contain at least one vertex in A_i (which will locate b°_i from $b^{\star}_i$) such that the union of these vertices resolves all pairs of the form $(c^{\circ}_j, c^{\star}_j)$, and hence corresponds to a satisfying assignment of ψ.

Proof of Theorem 2. Assume there exists an algorithm, say $\mathcal{A}$, that takes as input an instance (G, k) of Locating-Dominating Set and correctly concludes whether it is a Yes-instance in time $2^{o(k^2)} \cdot |V(G)|^{\mathcal{O}(1)}$. Consider algorithm $\mathcal{B}$ that takes as input an instance ψ of 3-SAT, uses the reduction above to get an equivalent instance (G, k) of Locating-Dominating Set, and then uses $\mathcal{A}$ as a subroutine. The correctness of algorithm $\mathcal{B}$ follows from the correctness of the reduction and of algorithm $\mathcal{A}$. From the description of the reduction and the fact that $k = \sqrt{n}$, the running time of algorithm $\mathcal{B}$ is $2^{\mathcal{O}(\sqrt{n})} + 2^{o(k^2)} \cdot (2^{\mathcal{O}(\sqrt{n})})^{\mathcal{O}(1)} = 2^{o(n)}$. This, however, contradicts the ETH. Hence, Locating-Dominating Set does not admit an algorithm with running time $2^{o(k^2)} \cdot |V(G)|^{\mathcal{O}(1)}$ unless the ETH fails.

For the second part of Theorem 2, assume that such a kernelization algorithm exists. Consider the following algorithm for 3-SAT. Given a 3-SAT formula on n variables, it uses the above reduction to get an equivalent instance of (G, k) such that $|V(G)| = 2^{\mathcal{O}(\sqrt{n})}$ and $k = \mathcal{O}(\sqrt{n})$. Then, it uses the assumed kernelization algorithm to construct an equivalent instance (H, k') such that H has $2^{o(k)}$ vertices and $k' \leq k$. Finally, it uses a brute-force algorithm, running in time $|V(H)|^{\mathcal{O}(k')}$, to determine whether the reduced instance, equivalently the input instance of 3-SAT, is a Yes-instance. The correctness of the algorithm follows from the correctness of the respective algorithms and our assumption. The total running time of the algorithm is $2^{\mathcal{O}(\sqrt{n})} + (|V(G)| + k)^{\mathcal{O}(1)} + |V(H)|^{\mathcal{O}(k')} = 2^{\mathcal{O}(\sqrt{n})} + (2^{\mathcal{O}(\sqrt{n})})^{\mathcal{O}(1)} + (2^{o(\sqrt{n})})^{\mathcal{O}(\sqrt{n})} = 2^{o(n)}$. This, however, contradicts the ETH. ◀

5 Test Cover Parameterization by the Solution Size

In this section, we present a reduction that is very close to the reduction used in the proof of Theorem 2 to prove Theorem 3.

For notational convenience, instead of specifying an instance of Test Cover, we specify the auxiliary graph as mentioned in the definition. The reduction takes as input an instance ψ, with n variables and m clauses, of 3-SAT and returns a reduced instance $(G, \langle R, B\rangle, k)$ of Test Cover and $k = \mathcal{O}(\log(n) + \log(m)) = \mathcal{O}(\log(n))$, using the sparcification lemma [44]. The reduction constructs graph G as follows.

- The reduction adds some dummy variables to ensure that $n = 2^{2q}$ for some positive integer q which is power of 2. This ensures that $r = \log_2(n) = 2q$ and $s = \frac{n}{r}$ both are integers. It partitions the variables of ψ into r many *buckets* $X_1, X_2, \ldots, X_r$ such that each bucket contains s many variables. Let $X_i = \{x_{i,j} \mid j \in [s]\}$ for all $i \in [r]$.
- For every X_i, the reduction constructs a set A_i of 2^s many red vertices, that is, $A_i = \{a_{i,\ell} \mid \ell \in [2^s]\}$. Each vertex in A_i corresponds to a unique assignment of the variables in X_i. Moreover, let $A = \cup_{i=1}^{r} A_i$.
- Corresponding to each X_i, let the reduction add a blue vertex b_i and the edges $(b_i, a_{i,\ell})$ for all $i \in [r]$ and $\ell \in [2^s]$. Let $B = \{b_i \mid i \in [r]\}$.
- For every clause C_j, the reduction adds a pair of blue vertices $c_j^\circ, c_j^\star$. For a vertex $a_{i,\ell} \in A_i$ with $i \in [r]$, and $\ell \in [2^s]$, if the assignment corresponding to vertex $a_{i,\ell}$ satisfies the clause C_j, then the reduction adds the edge $(a_{i,\ell}, c_j^\circ)$. Let $C = \{c_j^\circ, c_j^\star \mid j \in [m]\}$.
- The reduction adds a bit-representation gadget to locate set C. However, it adds the vertices in such a way that for any pair $c_j^\circ, c_j^\star$, the supplementary vertices adjacent to them are identical.
  - The reduction sets $p := \lceil \log(m) \rceil + 1$ and for every $i \in [p]$, it adds two vertices, a red vertex $z_{i,1}$ and a blue vertex $z_{i,2}$, and edge $(z_{i,1}, z_{i,2})$.
  - For every integer $j \in [m]$, let $\texttt{bit}(j)$ denote the binary representation of j using p bits. Connect $c_j^\circ, c_j^\star \in C$ with $z_{i,1}$ if the i^{th} bit in $\texttt{bit}(j)$ is 1.
  - It add two vertices $z_{0,1}$ and $z_{0,2}$, and edge $(z_{0,1}, z_{0,2})$. It also makes every vertex in C adjacent with $z_{0,1}$. Let $\texttt{bit-rep}(C)$ be the collection of all the vertices added in this step.
- The reduction adds an isolated blue vertex b_0.

This completes the construction. The reduction sets $k = r + \frac{1}{2}|\texttt{bit-rep}(C)| = \mathcal{O}(\log(n)) + \mathcal{O}(\log(m)) = \mathcal{O}(\log(n))$, and returns $(G, \langle R, B\rangle, k)$ as an instance of Test Cover. We refer to Figure 3 for an illustration.

We present a brief overview of the proof of correctness for the backward direction ($\Leftarrow$). Suppose R' is a set of tests of the graph G of order at most k. Since b_0 is an isolated blue vertex of G, it implies that the set R' dominates and locates every pair of vertices in $B \setminus \{b_0\}$. The blue vertices in $\texttt{bit-rep}(C)$ are pendant vertices that are adjacent with red vertices in $\texttt{bit-rep}(C)$. Hence, all the red vertices in $\texttt{bit-rep}(C)$ are in R'. The remaining r many vertices need to locate vertices in pairs $(c_j^\circ, c_j^\star)$, for every $j \in [m]$, which have the same neighbourhood in $\texttt{bit-rep}(C)$. To do so, note that the only vertices adjacent to c_j° and *not* to $c_j^\star$ are in A_i and corresponds to an assignment satisfying clause C_j. Hence, for every $j \in [m]$, the set R' should contain at least one vertex $\{a_{i,\ell}\}$ in order to locate c_j°, $c_j^\star$, where $(a_{i,\ell}, c_j^\circ)$ is an edge for some $i \in [r]$ and $\ell \in [2^s]$. Moreover, in order to dominate the vertices of B, for each $i \in [r]$, the set R' must have a vertex from each A_i. Hence, the set R' is forced to contain *exactly* one vertex from each A_i. Concatenating the assignments corresponding to each $a_{i,\ell}$ in R', we thus obtain a satisfying assignment of ψ. Proof of Theorem 3 follows from the arguments that are standard to proving such lower bounds.

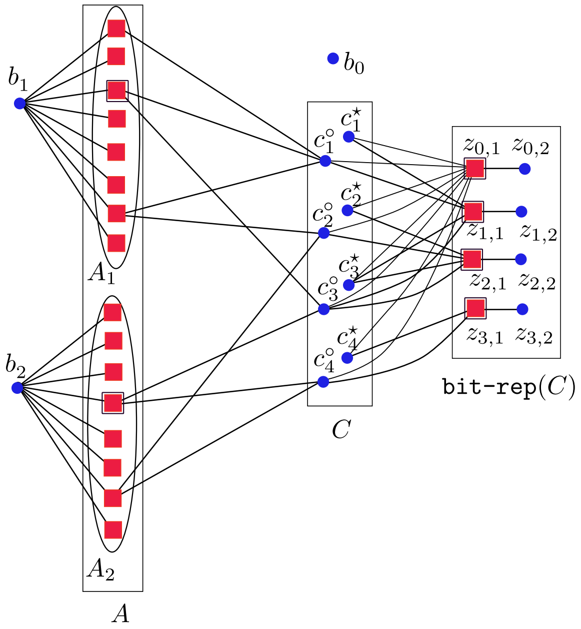


Figure 3 An illustrative example of the graph constructed by the reduction in Section 5. Red (squared) nodes denote the tests whereas blue (filled circle) nodes the elements.

6 Conclusion

We presented several results that advance our understanding of the algorithmic complexity of Locating-Dominating Set and Test Cover, which we showed to have very interesting and rare parameterized complexities. Moreover, we believe the techniques used in this article can be applied to other identification problems to obtain relatively rare conditional lower bounds. The process of establishing such lower bounds boils down to designing *bit-representation gadgets* and *set-representation gadgets* for the problem in question.

Apart from the broad question of designing such lower bounds for other identification problems, we mention an interesting problem left open by our work. Can our tight double-exponential lower bound for Locating-Dominating Set parameterized by treewidth/treedepth be applied to the feedback vertex set number? The question could also be studied for other related parameters.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. *ACM Trans. Comput. Theory*, 11(3):18:1–18:22, 2019. doi:10.1145/3319909.

2 Ian Anderson. *Combinatorics of Finite Sets*. Oxford University Press, 1987.

3 Gabriela R. Argiroffo, Silvia M. Bianchi, Yanina Lucarini, and Annegret Katrin Wagler. Linear-time algorithms for three domination-based separation problems in block graphs. *Discret. Appl. Math.*, 281:6–41, 2020. doi:10.1016/J.DAM.2019.08.001.

4 László Babai. On the complexity of canonical labelling of strongly regular graphs. *SIAM J. Comput.*, 9(1):212–216, 1980. doi:10.1137/0209018.

5 Florian Barbero, Lucas Isenmann, and Jocelyn Thiebaut. On the distance identifying set meta-problem and applications to the complexity of identifying problems on graphs. *Algorithmica*, 82(8):2243–2266, 2020. doi:10.1007/S00453-020-00674-X.

6 Manu Basavaraju, Mathew C. Francis, M. S. Ramanujan, and Saket Saurabh. Partially polynomial kernels for set cover and test cover. *SIAM J. Discret. Math.*, 30(3):1401–1423, 2016. doi:10.1137/15M1039584.

7 Piotr Berman, Bhaskar DasGupta, and Ming-Yang Kao. Tight approximability results for test set problems in bioinformatics. *J. Comput. Syst. Sci.*, 71(2):145–162, 2005. doi:10.1016/J.JCSS.2005.02.001.

8 Ivan Bliznets and Markus Hecher. Tight double exponential lower bounds. In Xujin Chen and Bo Li, editors, *Theory and Applications of Models of Computation - 18th Annual Conference, TAMC 2024, Hong Kong, China, May 13-15, 2024, Proceedings*, volume 14637 of *Lecture Notes in Computer Science*, pages 124–136. Springer, 2024. doi:10.1007/978-981-97-2340-9_11.

9 John A Bondy. Induced subsets. *Journal of Combinatorial Theory, Series B*, 12(2):201–202, 1972.

10 Koen M. J. De Bontridder, Bjarni V. Halldórsson, Magnús M. Halldórsson, Cor A. J. Hurkens, Jan Karel Lenstra, R. Ravi, and Leen Stougie. Approximation algorithms for the test cover problem. *Math. Program.*, 98(1-3):477–491, 2003. doi:10.1007/S10107-003-0414-6.

11 Márcia R. Cappelle, Guilherme de C. M. Gomes, and Vinícius Fernandes dos Santos. Parameterized algorithms for locating-dominating sets. *CoRR*, abs/2011.14849, 2020. arXiv:2011.14849.

12 Márcia R. Cappelle, Guilherme C. M. Gomes, and Vinícius Fernandes dos Santos. Parameterized algorithms for locating-dominating sets. In Carlos E. Ferreira, Orlando Lee, and Flávio Keidi Miyazawa, editors, *Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium, LAGOS 2021, Online Event / São Paulo, Brazil, May 2021*, volume 195 of *Procedia Computer Science*, pages 68–76. Elsevier, 2021. doi:10.1016/J.PROCS.2021.11.012.

13 Dipayan Chakraborty, Anni Hakanen, and Tuomo Lehtilä. The $n/2$-bound for locating-dominating sets in subcubic graphs, 2024. arXiv:2406.19278.

14 Jérémie Chalopin, Victor Chepoi, Fionn Mc Inerney, and Sébastien Ratel. Non-clashing teaching maps for balls in graphs. *CoRR*, abs/2309.02876, 2023. doi:10.48550/arXiv.2309.02876.

15 L. Sunil Chandran, Davis Issac, and Anreas Karrenbauer. On the parameterized complexity of biclique cover and partition. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016*, volume 63 of *LIPIcs*, pages 11:1–11:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.IPEC.2016.11.

16 Emmanuel Charbit, Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in bipartite graphs: bounds, extremal cardinalities, complexity. *Advances in Mathematics of Communication*, 2(4):403–420, 2008. doi:10.3934/AMC.2008.2.403.

17 Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in (bipartite) planar graphs. *Eur. J. Comb.*, 29(5):1353–1364, 2008. doi:10.1016/J.EJC.2007.05.006.

18 Bogdan S. Chlebus and Sinh Hoa Nguyen. On finding optimal discretizations for two attributes. In *Proceedings of the First International Conference on Rough Sets and Current Trends in Computing*, volume 1424, pages 537–544, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/3-540-69115-4_74.

19 Vasek Chvátal. Mastermind. *Combinatorica*, 3(3):325–329, 1983. doi:10.1007/BF02579188.

20 C. Colbourn, P. J. Slater, and L. K. Stewart. Locating-dominating sets in series-parallel networks. *Congressus Numerantium*, 56:135–162, 1987.

21 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

22 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

23 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/S002249910009.

24 Robert Crowston, Gregory Z. Gutin, Mark Jones, Gabriele Muciaccia, and Anders Yeo. Parameterizations of test cover with bounded test sizes. *Algorithmica*, 74(1):367–384, 2016. doi:10.1007/S00453-014-9948-7.

25 Robert Crowston, Gregory Z. Gutin, Mark Jones, Saket Saurabh, and Anders Yeo. Parameterized study of the test cover problem. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 2012. doi:10.1007/978-3-642-32589-2_27.

26 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

27 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016. doi:10.1137/130947076.

28 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics.* Springer, 2012. URL: https://dblp.org/rec/books/daglib/0030488.bib.

29 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. doi:10.1007/978-3-642-02927-1_32.

30 Johannes Klaus Fichte, Markus Hecher, Michael Morak, Patrick Thier, and Stefan Woltran. Solving projected model counting by utilizing treewidth and its limits. *Artif. Intell.*, 314:103810, 2023. doi:10.1016/J.ARTINT.2022.103810.

31 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting treewidth for projected model counting and its limits. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Proc.*, volume 10929 of *Lecture Notes in Computer Science*, pages 165–184. Springer, 2018. doi:10.1007/978-3-319-94144-8_11.

32 Jacob Focke, Fabian Frei, Shaohua Li, Dániel Marx, Philipp Schepper, Roohani Sharma, and Karol Wegrzycki. Hitting meets packing: How hard can it be? In *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 55:1–55:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.55.

33 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

34 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing.* Cambridge University Press, 2019.

35 Florent Foucaud. Decision and approximation complexity for identifying codes and locating-dominating sets in restricted graph classes. *J. Discrete Algorithms*, 31:48–68, 2015. doi:10.1016/J.JDA.2014.08.004.

36 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Problems in NP can admit double-exponential lower bounds when parameterized by treewidth or vertex cover. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 66:1–66:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.66.

37 Florent Foucaud, George B. Mertzios, Reza Naserasr, Aline Parreau, and Petru Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. I. bounds. *Theoretical Computer Science*, 668:43–58, 2017. doi:10.1016/J.TCS.2017.01.006.

38 Florent Foucaud, George B Mertzios, Reza Naserasr, Aline Parreau, and Petru Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. II. algorithms and complexity. *Algorithmica*, 78(3):914–944, 2017. doi:10.1007/S00453-016-0184-1.

39 M. R. Garey and David S. Johnson. *Computers and Intractability – A guide to NP-completeness.* W.H. Freeman and Company, 1979.

40 Sylvain Gravier, Ralf Klasing, and Julien Moncel. Hardness results and approximation algorithms for identifying codes and locating-dominating codes in graphs. *Algorithmic Oper. Res.*, 3(1), 2008. URL: http://journals.hil.unb.ca/index.php/AOR/article/view/2808.

41 Gregory Z. Gutin, Gabriele Muciaccia, and Anders Yeo. (non-)existence of polynomial kernels for the test cover problem. *Inf. Process. Lett.*, 113(4):123–126, 2013. doi:10.1016/J.IPL.2012.12.008.

42 Tesshu Hanaka, Noleen Köhler, and Michael Lampis. Core stability in additively separable hedonic games of low treewidth, 2024. arXiv:2402.10815, doi:10.48550/arXiv.2402.10815.

43 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

44 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

45 Klaus Jansen, Kim-Manuel Klein, and Alexandra Lassota. The double exponential runtime is tight for 2-stage stochastic ILPs. *Math. Program.*, 197:1145–1172, 2023. doi:10.1007/S10107-022-01837-0.

46 D. Jean and A Lobstein. Watching systems, identifying, locating-dominating and discriminating codes in graphs: a bibliography, 2024. Published electronically at https://dragazo.github.io/bibdom/main.pdf.

47 Dusan Knop, Michal Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *ACM Trans. Comput. Theory*, 12(3):19:1–19:19, 2020. doi:10.1145/3397484.

48 Lukasz Kowalik, Alexandra Lassota, Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Detecting points in integer cones of polytopes is double-exponentially hard. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 279–285, 2024. doi:10.1137/1.9781611977936.25.

49 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: The easy kernel is essentially tight. *ACM Trans. Algorithms*, 12(3):40:1–40:16, 2016. doi:10.1145/2832912.

50 M. Künnemann, F. Mazowiecki, L. Schütze, H. Sinclair-Banks, and K. Węgrzycki. Coverability in VASS Revisited: Improving Rackoff's Bound to Obtain Conditional Optimality. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 131:1–131:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.ICALP.2023.131.

51 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to Courcelle's theorem. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018*, volume 10929 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018. doi:10.1007/978-3-319-94144-8_15.

52 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

53 Daniel Lokshtanov, Saket Saurabh, Subhash Suri, and Jie Xue. An ETH-tight algorithm for multi-team formation. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, volume 213 of *LIPIcs*, pages 28:1–28:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FSTTCS.2021.28.

54 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 28:1–28:15, 2016. doi:10.4230/LIPICS.ICALP.2016.28.

55 Bernard M. E. Moret and Henry D. Shapiro. On minimizing a set of tests. *SIAM Journal on Scientific and Statistical Computing*, 6(4):983–1003, 1985.

56 Tobias Müller and Jean-Sébastien Sereni. Identifying and locating-dominating codes in (random) geometric networks. *Comb. Probab. Comput.*, 18(6):925–952, 2009. doi:10.1017/S0963548309990344.

57 Marcin Pilipczuk and Manuel Sorge. A double exponential lower bound for the distinct vectors problem. *Discret. Math. Theor. Comput. Sci.*, 22(4), 2020. doi:10.23638/DMTCS-22-4-7.

58 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

59 N.S.V. Rao. Computational complexity issues in operative diagnosis of graph-based systems. *IEEE Transactions on Computers*, 42(4):447–457, 1993. doi:10.1109/12.214691.

60 Alfred Rényi. On random generating elements of a finite boolean algebra. *Acta Scientiarum Mathematicarum Szeged*, 22:75–81, 1961.

61 Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. *Inf. Comput.*, 281:104812, 2021. doi:10.1016/J.IC.2021.104812.

62 András Sebö and Eric Tannier. On metric generators of graphs. *Mathematics of Operations Research*, 29(2):383–393, 2004. doi:10.1287/MOOR.1030.0070.

63 Peter J. Slater. Domination and location in acyclic graphs. *Networks*, 17(1):55–64, 1987. doi:10.1002/net.3230170105.

64 Peter J. Slater. Dominating and reference sets in a graph. *Journal of Mathematical and Physical Sciences*, 22(4):445–455, 1988.

65 Jukka Suomela. Approximability of identifying codes and locating-dominating codes. *Inf. Process. Lett.*, 103(1):28–33, 2007. doi:10.1016/J.IPL.2007.02.001.

66 P. Tale. Double exponential lower bound for telephone broadcast, 2024. arXiv:2403.03501, doi:10.48550/arXiv.2403.03501.

67 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

Revisit the Scheduling Problem with Calibrations

Lin Chen
Department of Computer Science, Zhejiang University, China

Yixiong Gao
Department of Computer Science, City University of Hong Kong, Kowloon, China

Minming Li
Department of Computer Science, City University of Hong Kong, Kowloon, China

Guohui Lin
Department of Computing Science, University of Alberta, Edmonton, Canada

Kai Wang
Department of Computer Science, City University of Hong Kong, Kowloon, China

Abstract

The research about scheduling with calibrations was initiated from the Integrated Stockpile Evaluation (ISE) program which tests nuclear weapons periodically. The tests for these weapons require calibrations that are expensive in the monetary sense. This model has many industrial applications where the machines need to be calibrated periodically to ensure high-quality products, including robotics and digital cameras. In 2013, Bender et al. (SPAA '13) proposed a theoretical framework for the ISE problem. In this model, a machine can only be trusted to run a job when it is calibrated and the calibration remains valid for a time period of length T, after which it must be recalibrated before running more jobs. The objective is to find a schedule that completes all jobs by their deadlines and minimizes the total number of calibrations. In this paper, we study the scheduling problem with calibrations on multiple parallel machines where we consider unit-time processing jobs with release times and deadlines. We propose a dynamic programming algorithm with polynomial running time when the number of machines is constant. Then, we propose another dynamic programming approach with polynomial running time when the length of the calibrated period is constant. Also, we propose a PTAS, that is, for any constant $\epsilon > 0$, we give a $(1 + \epsilon)$ - approximation solution with m machines.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Approximation Algorithm, Scheduling, Calibration, Resource Augmentation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.20

1 Introduction

The original motivation for scheduling with calibrations came directly from the Integrated Stockpile Evaluation (ISE) program which tests nuclear weapons periodically [6]. The tests for these weapons require calibrations that are expensive. This motivation can be extended to the scenarios where the machines need to be calibrated periodically to ensure high-quality products, such as robotics and digital cameras [17, 4, 27]. In this model, a machine must be calibrated before it runs a job. When the machine is calibrated at time t, it stays calibrated for a time period of length T, after which it must be recalibrated to run more jobs. We refer to the time interval $[t, t + T]$ as the *calibration interval* and no job can be started or be processed on a machine outside the times sitting in a calibration interval. In the ideal model, calibrating a machine is instantaneous, meaning that the machine can run a job immediately after being calibrated and the machine can switch from uncalibrated to calibrated status instantaneously. The objective is to assign the jobs to the machines using the minimum number of calibrations.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 20; pp. 20:1–20:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Methodologies have been introduced about performing a calibration in different scenarios [20, 23, 26], as well as determining the time period of a calibration [16, 18]. On the other hand, there is quite a lot of research work about scheduling algorithms [5]. In 2013, Bender et al. [3] proposed a theoretical framework for scheduling with calibrations. They considered unit-time jobs with release times and deadlines, aiming at minimizing the number of calibrations. In the single-machine setting, they proposed a greedy, optimal, polynomial-time algorithm called *Lazy-Binning* where the algorithm delays the start of a calibration interval for as long as possible, until delaying it further would make it impossible to find a feasible schedule. For the multiple machine setting, they proposed the Lazy-Binning algorithm on multiple machines and showed it is a 2-approximation algorithm, while the complexity status still remains open.

Later, Fineman and Sheridan [12] considered the non-preemptive jobs with non-unit processing times and generalized the problem with resource-augmentation [15] in the sense that an approximate solution can be obtained if we speed up the machines and/or have more machines. They worked on multiple machine setting without preemption, showed the relationship of the problem with the classical machine-minimization problem [19] and proved that if there is an s-speed α-approximation algorithm for the machine-minimization problem, then it would give an s-speed $O(\alpha)$-machine $O(\alpha)$-approximation solution for the calibration minimization problem.

Angel et al. [2] developed different results on several generalizations of this problem. They considered the jobs of non-unit processing times on a single machine with preemption, and proposed an optimal greedy algorithm, which extends the idea of the Lazy Binning algorithm. Also, they extended the model to allow many types of calibrations where different types of calibrations have different interval lengths and costs, and proved that the problem is NP-hard. At last, they considered a more realistic case where calibrating a machine takes a fixed amount of time, and proposed a dynamic programming approach to solve the problem. Chau et al. [7] showed a 3-approximation polynomial time algorithm when jobs have unit processing times. Also, they showed a $(3/(1-\varepsilon))$-approximation pseudo-polynomial time algorithm and a $(18/(1-\varepsilon))$-approximation polynomial time algorithm for the arbitrary processing time case. Chau et al. [9] showed that the scheduling problem with batch calibrations can be solved in polynomial time. Also, they proposed some fast approximation algorithms for several special cases. Chen and Zhang [11] considered online scheduling with calibration while calibrating a machine will require certain time units. They gave an asymptotically optimal algorithm for this problem when all the jobs have unit processing times, and improved the competitive ratio for the special case that calibrating a machine is instantaneous.

While the above works are about minimizing the number of calibrations, Chau et al. [8] worked on the trade-off between weighted flow time and calibration cost for unit-time jobs. They integrated the objective function with these two criteria, and gave several online approximation results on different settings and also a dynamic programming method for the offline problem. Wang [24] worked on the scheduling of minimizing total time slot cost with calibration requirements. They considered jobs of identical processing times, and proposed three different dynamic programs for different scenarios. Chen et al. [10] studied the scheduling problem with multiple types of calibrations and proposed constant approximation algorithms for several types of calibrations.

In this paper, we work on the original problem proposed by Bender et al. [3] in which jobs have unit processing times with release times and deadlines. We abbreviate this problem as $P|r_j, p_j = 1, d_j, T|\#$*Calibrations with machine calibrations*, where we adopt the classical three-field notation in scheduling of Graham *et al.* [13]. We propose two dynamic programming approaches to solve the problem with polynomial running time when (i) the number of machines is constant, (ii) or the valid length of a calibration is constant. Moreover, when the number of machine and the valid length of a calibration are both inputs, we present a PTAS. Our results are summarized in Table 1. It is worth noting that the currently best result for this problem is a 2-approximation algorithm by Bender et al. [3], and the complexity status still remains open.

Table 1 A summary of the results of scheduling with calibrations on multiple machines.

Algorithm	Approximation Ratio	Time Complexity	Remark
Section 3	1	$O(n^{8+6m})$	m is constant, T is input
Section 4	1	$O(n^8 m^{3T})$	m is input, T is constant
Section 5	$1+\epsilon$	$O(n^{17+18\lceil 1/\epsilon \rceil})$	Both m, T are input
Bender et al. [3]	2	Poly(n, m)	Lazy-binning approach

Transforming a dynamic programming formulation into a PTAS (polynomial time approximation scheme) has been studied decades ago [14, 21]. Woeginger [25] proposed a general technique and gave some conditions to identify whether a dynamic programming formulation could be transformed into an FPTAS. Schuurman and Woeginger [22] summarized the methods into three main categories, structuring the input, structuring the output and structuring the execution of an algorithm.

For our problem, it is not straightforward to directly apply Schuurman and Woeginger's [22] method to get an FPTAS. Therefore, in this paper, we show a different approach to transform the dynamic program formulation into a PTAS, which lies in the category of structuring the output. In Section 2 we present the structure of an optimal schedule of this problem. Then in Section 3 we propose a dynamic programming approach to solve the problem. In Section 4 we propose another dynamic programming approach for the case when T is constant. In Section 5 we give a PTAS algorithm. We conclude our results in Section 6.

2 Formulation

We are given a set J of n jobs, where each job $j \in J$ has release time r_j, deadline d_j and processing time $p_j = 1$. We have m identical parallel machines which can be trusted to run a job only when calibrated. The calibration remains valid for a time period of length T. The objective is to find a schedule that completes all jobs before their deadlines such that the number of calibrations is minimized. We assume that all the input are non-negative integers.

A feasible solution includes the schedule of calibrations (i.e., when to start a calibration on each machine) and the schedule of jobs (i.e., when and on which machine to start a job). We assume all the inputs are integers and both calibrations and jobs should start at times which are also integers. We denote the time interval $(t-1, t]$ as *time slot* t and t is called *active* if some job is scheduled in this time slot. We sort the jobs by non-decreasing order of their deadlines, and non-decreasing order of release times if two or more jobs have the same deadline. The jobs are indexed from 1 to n, and any job $j \in J$ has index $j \in [1, n]$. As a matter of fact, once the schedule of calibrations is fixed, the schedule of jobs can be obtained by applying the classical *Earliest-Deadline-First* (EDF) scheduling algorithm, in which for any time slot, the job of the earliest deadline has the highest priority to be considered to schedule whenever a machine is available (i.e., calibrated).

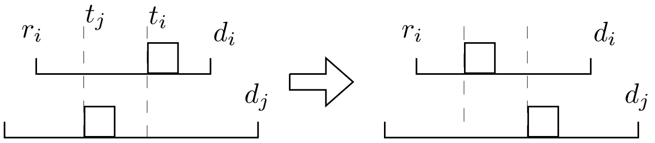


Figure 1 An illustration for Lemma 1.

▶ **Lemma 1** (EDF). *There exists an optimal schedule such that for any two jobs i, j with $i < j$, if $r_i \le t_j$ then $t_i \le t_j$ where t_i, t_j are the corresponding starting times of job i and j in the optimal schedule respectively.*

Proof. As job i has smaller index than job j, i.e., $i < j$, we have $d_i \le d_j$. Suppose $t_i > t_j$ in the optimal schedule, and then we have $r_i \le t_j < t_i < d_i \le d_j$, which implies that by swapping the schedule of the two jobs i and j (schedule job i at time t_j and job j at time t_i), the new schedule is still feasible and follows EDF scheduling policy. In order to prove that another optimal schedule can be obtained after a finite number of the above swapping process, we only need to show that a certain value of the schedule decreases after the swapping process. Especially, after swapping, the value $(i - t_i)^2 + (j - t_j)^2$ strictly decreases since $(i - t_i)^2 + (j - t_j)^2 > (i - t_j)^2 + (j - t_i)^2$. Therefore, we will eventually obtain an optimal schedule satisfying the statement. ◀

▶ **Definition 2.** *Let* $\Psi = \bigcup_{j \in J, s \in [0,n]} \{d_j - s\}$, $\Phi = \bigcup_{j \in J, t \in \Psi, s \in [0,n]} \{r_j + s, t + s\}$ *and* $\Phi(j) = \{t \mid r_j < t \le d_j,\ t \in \Phi\}$, $\forall j \in J$.

One would find that $|\Psi| = O(n^2)$. Also $|\Phi| = O(n^2)$ since for each $t \in \Psi$ we have $t = d_j - s$ for some $j \in J, s \in [0, n]$ and the number of possible values of $d_j - s + s'$ for $s' \in [0, n]$ is bounded by $O(n^2)$. The following lemma is from the work by Angel et al. [1].

▶ **Lemma 3** ([1]). *There always exists an optimal schedule such that*

i.) *each calibration starts at a time in* Ψ.

ii.) $\forall j \in J$, *job* j *finishes at a time in* $\Phi(j)$.

3 Dynamic Programming Approach

In this section we introduce a dynamic programming approach to solve the problem. Assume that the calibrations on the same machine never overlap with each other and we look for the optimal solution which follows EDF scheduling policy. We focus on the problem within a specific time interval $[t_1, t_2)$ and consider the jobs that are released during this interval, where t_1, t_2 are the possible job completion times. Lemma 1 shows a very important property that once job j is scheduled at time slot t in the optimal solution, the remaining jobs (whose index is less than j) could be partitioned into two groups such that they must be scheduled during time intervals $[t_1, t)$ and $[t, t_2)$ in the optimal solution, respectively (more details in later analysis). Therefore, we could split the problem into sub-problems. In the sub-problem, we need to mark the calibrations that cross the boundary of the time interval $[t_1, t_2)$, where we use vectors defined in the following.

Given time slot t, some machines may be calibrated at slot t and others may not. We use notation NUL to represent the situation that the machine is not calibrated at some time slot (NUL represents *NULL* in programming). In order to mark the calibrations on each machine that cover time slot t (there could be at most m such calibrations), we use a vector $\gamma = \langle \gamma_1, \gamma_2, ..., \gamma_m \rangle$ to represent the starting times of these calibrations where

$\gamma_k \in \{\text{NUL}\} \cup (\Psi \cap [t-T, t))$ indicates the starting time of the calibration on machine k. Let $\Gamma(t) = \{\langle \gamma_1, \gamma_2, ..., \gamma_m \rangle \mid \gamma_k \in \{\text{NUL}\} \cup (\Psi \cap [t-T, t)),\ \forall k \in [1, m]\}$ be the set of all possible vectors, with respect to time slot t.

▶ **Definition 4.** *Let $J(j, t_1, t_2)$, $\forall j \in J$ be the subset of jobs whose index is at most j and release time is between t_1 and t_2, i.e., $J(j, t_1, t_2) = \{i \mid i \le j, r_i \in [t_1, t_2), i \in J\}$*

▶ **Definition 5.** *Let $f(j, t_1, t_2, q, \boldsymbol{u}, \boldsymbol{v})$ be the minimum number of calibrations to schedule jobs $J(j, t_1, t_2)$ on m machines where $\boldsymbol{u} = \langle u_1, u_2, ..., u_m \rangle, \boldsymbol{u} - T \in \Gamma(t_1)$, $\boldsymbol{v} = \langle v_1, v_2, ..., v_m \rangle \in \Gamma(t_2)$, $t_1 \in \Phi$, $t_2 \in \Phi$, $q \in [0, m]$ on the condition that*

i.) *jobs $J(j, t_1, t_2)$ are only scheduled during time interval $[t_1, t_2)$.*
ii.) *time intervals $[t_1, u_k)$ and $[v_k, t_2)$ have already been calibrated on machine k, $\forall k \in [1, m]$.*
iii.) *q other jobs (not from $J(j, t_1, t_2)$) have already been assigned to time slot t_2.*

In the definition, we use vector $\boldsymbol{u}$ (resp. $\boldsymbol{v}$) to mark the calibration ending times (resp. starting times) of the calibrations that cross the boundary of interval $[t_1, t_2)$, i.e., covering time slot t_1 (resp. t_2), where $\boldsymbol{u} - T \in \Gamma(t_1)$ (resp. $\boldsymbol{v} \in \Gamma(t_2)$). We use parameter q to reserve q machines at slot t_2 in order to schedule the jobs (not from $J(j, t_1, t_2)$) that are assigned to slot t_2.

We consider the starting time of job j and suppose job j is scheduled at time slot t in the optimal schedule (refer to Figure 2). If a job from $J(j-1, t_1, t_2)$ is released before t, then it must be scheduled before (or at) time slot t by Lemma 1. Therefore, the remaining jobs $J(j-1, t_1, t_2)$ can be partitioned into two groups: jobs $J(j-1, t_1, t)$ and jobs $J(j-1, t, t_2)$. For jobs $J(j-1, t_1, t)$, they will not be scheduled after time t as argued, and for jobs $J(j-1, t, t_2)$ they cannot be scheduled before t because of the job release time. Hence the original problem could be divided into two sub-problems. Moreover, we have to enumerate the calibrations (i.e., the calibration starting times) on each machine that cover time slot t in the optimal schedule, in order to schedule job j at time slot t. Specifically, we use vector $\boldsymbol{x} = \langle x_1, x_2, ..., x_m \rangle \in \Gamma(t)$ to indicate the starting times of the calibrations on all machines and correspondingly $\boldsymbol{y} = \langle y_1, y_2, ..., y_m \rangle = \boldsymbol{x} + T$ as the calibration ending times.

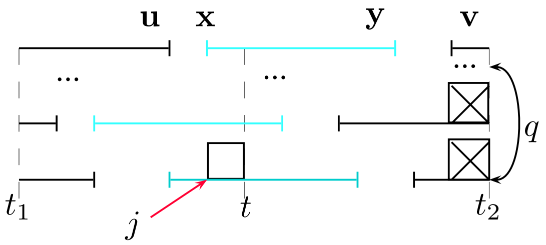


Figure 2 An illustration for Proposition 7.

In the following, we define the operations that is related to NUL.

▶ **Definition 6.** *For any $x \in \mathbb{R}$, we define the min and max functions $\min\{\text{NUL}, x\}$, $\min\{x, \text{NUL}\}, \max\{\text{NUL}, x\}$, $\max\{x, \text{NUL}\}$ to be x, the operations $x + \text{NUL}, x - \text{NUL}, \text{NUL} - \text{NUL}, \min\{\text{NUL}, \text{NUL}\}$ to be NUL, and the intervals $[x, \text{NUL}), [\text{NUL}, x)$ to be $\emptyset$.*

We define min (or max, analogously) function on two vectors $\boldsymbol{\gamma}, \boldsymbol{\lambda}$ to be $\boldsymbol{\beta} = \min\{\boldsymbol{\gamma}, \boldsymbol{\lambda}\}$ where $\beta_k = \min\{\gamma_k, \lambda_k\}$, $\forall k \in [1, m]$ (recall that γ_k or λ_k might be NUL). We define operator $+$ (or $-$, analogously) on a vector $\boldsymbol{\lambda}$ and a number x to be $\boldsymbol{\beta} = \boldsymbol{\lambda} + x$ where $\beta_k = \lambda_k + x$. And we define operator $<$ (or $\le, >, \ge$, analogously) to be $\boldsymbol{\beta} = (\boldsymbol{\lambda} < x)$ where $\beta_k = \lambda_k$ if $\lambda_k \ne \text{NUL}, \lambda_k < x$ and otherwise $\beta_k = \text{NUL}$.

Let function $\delta(\boldsymbol{\lambda}) = \sum_{\lambda_k \neq \text{NUL}, k\in[1,m]} 1$ *on vector* $\boldsymbol{\lambda}$ *be the function that indicates the number of real values in vector* $\boldsymbol{\lambda}$.

▶ **Proposition 7.** *For the case* $J(j,t_1,t_2) = \emptyset$*, we set* $f(j,t_1,t_2,q,\boldsymbol{u},\boldsymbol{v})$ *to be* 0 *if at least* q *machines are calibrated at time slot* t_2 *providing* $\boldsymbol{u}$ *and* $\boldsymbol{v}$*, and otherwise* ∞*. If* $j \notin J(j,t_1,t_2)$ *we have* $f(j,t_1,t_2,q,\boldsymbol{u},\boldsymbol{v}) = f(j-1,t_1,t_2,q,\boldsymbol{u},\boldsymbol{v})$*. If* $j \in J(j,t_1,t_2)$ *and* $\Phi(j) \cap (t_1,t_2] = \emptyset$*, we have* $f(j,t_1,t_2,q,\boldsymbol{u},\boldsymbol{v}) = \infty$*. Otherwise we have* $f(j,t_1,t_2,q,\boldsymbol{u},\boldsymbol{v}) =$

$$\min_{t\in\Phi(j)\cap(t_1,t_2]} \begin{cases} \infty & , \textit{ if } t = t_2, q = m \\ f(j-1,t_1,t_2,q+1,\boldsymbol{u},\boldsymbol{v}) & , \textit{ if } t = t_2, 0 < q < m \\ \min \quad \delta(\boldsymbol{x}) & \\ \textit{cond.} & \\ +f(j-1,t_1,t,1,\boldsymbol{u},\boldsymbol{v}') & \\ +f(j-1,t,t_2,q,\boldsymbol{u}',\boldsymbol{v}) & , \textit{ if } t < t_2 \textit{ or } q = 0 \end{cases}$$

where cond. represents $\boldsymbol{x} \in \Gamma(t), \boldsymbol{y} = \boldsymbol{x} + T, \boldsymbol{u}' = \max\{\boldsymbol{y}, \boldsymbol{u} \geq t\}, \boldsymbol{v}' = \min\{\boldsymbol{x}, \boldsymbol{v} < t\}$.

Proof. For the base cases, if $J(j,t_1,t_2) = \emptyset$, no job needs to be scheduled, hence we only need to guarantee that at least q machines are calibrated at time slot t_2. If $j \notin J(j,t_1,t_2)$, we would have $J(j,t_1,t_2) = J(j-1,t_1,t_2)$. If $j \in J(j,t_1,t_2), \Phi(j) \cap (t_1,t_2] = \emptyset$, it is impossible to schedule job j during interval $(t_1,t_2]$.

In the dynamic programming equation, we allow calibrations to overlap with each other on the same machine, and we guarantee that for each time slot, the number of jobs that are assigned to this time slot is at most the number of machines that are calibrated during this time slot. Firstly, we try every possibility (specifically, job starting time) to schedule job j. Secondly, we follow the scheduling policy that whenever a time slot is active, we try every possibility of the calibrations (specifically, calibration starting times) on each machine that cover this time slot. Depending on the time slot t where job j is scheduled, we divide the analysis into three cases.

Case 1) $t = t_2,\ q = m$. In this case, there are already q jobs assigned to time slot t_2. Therefore it is infeasible to assign job j to time slot t_2.

Case 2) $t = t_2,\ 0 < q < m$. In this case, job j is assigned to time slot t_2. Since $q > 0$, i.e., some job is already scheduled at time slot t_2, by our approach the calibration decision on time slot t_2 is already made, which means that we do not need to enumerate again the calibrations that cover time slot t_2. Therefore we just schedule job j at slot t_2 and recurse to the sub-problem $f(j-1,t_1,t_2,q+1,\boldsymbol{u},\boldsymbol{v})$ where we reserve $q+1$ machines at time slot t_2.

Case 3) $t < t_2$ or $q = 0$. If $t < t_2$, time slot t is not yet active as we only reserve time slot t_2 for other jobs. If $t = t_2, q = 0$, no job is reserved at time slot t_2 by definition. Therefore, in this case time slot t is not yet active and we enumerate the calibrations that cover time slot t, i.e., vector $\boldsymbol{x} \in \Gamma(t)$. Once we fix time slot t and vector $\boldsymbol{x}$, we partition the remaining jobs $J(j-1,t_1,t_2)$ into two groups: jobs $J(j-1,t_1,t)$ and jobs $J(j-1,t,t_2)$. Because jobs in $J(j-1,t_1,t)$ will not be scheduled after time t in the optimal solution by Lemma 1 and jobs in $J(j-1,t,t_2)$ cannot be scheduled before t because of the job release time, the two groups of jobs must be scheduled during interval $[t_1,t)$ and $[t,t_2)$ respectively. The only issue left is to determine the calibrations that cross the boundary of the intervals. For the calibrations that intersect with interval $[t_1,t)$ and cover time slot t, they must come from the calibrations with starting time $\boldsymbol{x}$ or $\boldsymbol{v}$. Hence we define $\boldsymbol{v}' = \min\{\boldsymbol{x}, \boldsymbol{v} < t\}$ to include as many calibrated slots on each machine as possible for interval $[t_1,t)$ and define the first sub-problem to be $f(j-1,t_1,t,1,\boldsymbol{u},\boldsymbol{v}')$, in which we reserve one machine at time slot t for the schedule of job j. Note that $\boldsymbol{v}'$ follows the definition of the sub-problem, i.e., $\boldsymbol{v}' \in \Gamma(t)$. Symmetrically, we define vector $\boldsymbol{u'} = \max\{\boldsymbol{y}, \boldsymbol{u} \geq t\}$ to include as many calibrated slots on each machine as possible for interval $[t, t_2)$ and we have $\boldsymbol{u'} - T \in \Gamma(t)$. We define the sub-problem to be $f(j-1, t, t_2, q, \boldsymbol{u'}, \boldsymbol{v})$. As we have tried every possibility of time slot t, vector $\boldsymbol{x}$, at least one try is the same as the optimal solution. Hence, we obtain two partial schedules from two sub-problems respectively and then schedule job j at time slot t. Job j is feasible because the first sub-problem has reserved a time slot for job j at slot t by definition. ◀

Time Complexity. The dynamic program has table size $O(n^2|\Phi|^2|\Psi|^{2m})$. Constructing the solution from the sub-problems takes $O(|\Phi||\Psi|^m)$ steps. In total, the running time is $O(n^2|\Phi|^3|\Psi|^{3m}) = O(n^{8+6m})$. When m is constant, i.e., the number of machines is constant, our dynamic programming approach has polynomial running time.

4 When T is Constant

The algorithm in Section 3 is exponential on the number of machines (i.e., m). When m is input and T is constant, we introduce another dynamic programming approach in this section with polynomial running time. Since all jobs have unit processing times, the scheduling between two different time slots is independent once the calibration scheme and the starting time of each job are determined. That is, for a certain time slot t, a job processed on this slot and any two available (i.e., calibrated) machines on this slot, it makes no difference to schedule the job on either of the two machines for the schedule of any other time slot t'. This inspires us that we do not need to distinguish machines, but only need to distinguish calibrations with different starting times. When T is constant, we can use this to optimize the table's cardinality in the dynamic programming approach to a polynomial of n and m.

▶ **Definition 8.** *Assume that the calibrations on the same machine never overlap. To mark the status of calibrations at time slot t, we use a vector $\boldsymbol{c_t} = \langle c_{t,1}, c_{t,2}, ..., c_{t,T}\rangle$ to represent the numbers of calibrations distinguished by the number of remaining time slots that kept the machine calibrated, where $c_{t,k} \in \{0, 1, 2, \ldots, m\}$ indicates the number of calibrations that makes the machine available at time slot $t, t+1, \ldots, t+k-1$.*

We define function ι on vector $\boldsymbol{c_t}$ and an integer x as $\iota(\boldsymbol{c_t}, x) = \boldsymbol{c_{t+x}}$ where $c_{t+x,k} = c_{t,k-x}$ if $k - x \in [1, T]$, and otherwise 0, $\forall k \in [1, T]$. We define max function on two vectors $\boldsymbol{\gamma}, \boldsymbol{\lambda}$ to be $\boldsymbol{\beta} = \max\{\boldsymbol{\gamma}, \boldsymbol{\lambda}\}$ where $\beta_k = \max\{\gamma_k, \lambda_k\}$, $\forall k \in [1, T]$.

Let $\mathcal{C}(t) = \{\langle c_{t,1}, c_{t,2}, ..., c_{t,T}\rangle \mid c_{t,k} \in \{0, 1, 2, \ldots, m\},\ \forall k \in [1, T] \ \wedge\ \sum_{k=1}^{T} c_{t,k} \leq m\}$ be the set of all possible vectors, with respect to time slot t. One would find $|\mathcal{C}(t)| = O(m^T)$.

▶ **Definition 9.** *We define $f(j, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}})$ to be the minimum number of calibrations to schedule jobs $J(j, t_1, t_2)$ on m machines where $\boldsymbol{c_{t_1}} \in \mathcal{C}(t_1), \boldsymbol{c_{t_2}} \in \mathcal{C}(t_2)$, $t_1 \in \Phi$, $t_2 \in \Phi$, $q \in [0, m]$ on the condition that*

- **i.)** *jobs $J(j, t_1, t_2)$ are only scheduled during time interval $[t_1, t_2)$.*
- **ii.)** *machines have already been calibrated according to $\boldsymbol{c_{t_1}}$ and $\boldsymbol{c_{t_2}}$.*
- **iii.)** *q other jobs (not from $J(j, t_1, t_2)$) have already been assigned to time slot t_2.*

▶ **Proposition 10.** *For the case $J(j, t_1, t_2) = \emptyset$, we set $f(j, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}})$ to be 0 if there are at least q available machines on time slot t (i.e., $q \leq \sum_{k=1}^{T} c_{t_2,k}$), and otherwise ∞. If $j \notin J(j, t_1, t_2)$, we have $f(j, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}}) = f(j-1, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}})$. If $j \in J(j, t_1, t_2)$ and $\Phi(j) \cap (t_1, t_2] = \emptyset$, we have $f(j, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}}) = \infty$. Otherwise we have*

$$f(j, t_1, t_2, q, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}}) =$$

$$\min_{t \in \Phi(j) \cap (t_1, t_2]} \begin{cases} \infty & , \textit{if } t = t_2, q = m \\ f(j-1, t_1, t_2, q+1, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}}) & , \textit{if } t = t_2, 0 < q < m \\ \min\limits_{cond.} \quad \sum_{k=1}^{T} c_{t,k} & \\ +f(j-1, t_1, t, 1, \boldsymbol{c_{t_1}}, \dot{\boldsymbol{c}}_{\boldsymbol{t}}) & \\ +f(j-1, t, t_2, q, \ddot{\boldsymbol{c}}_{\boldsymbol{t}}, \boldsymbol{c_{t_2}}) & , \textit{if } t < t_2 \textit{ or } q = 0 \end{cases}$$

where cond. stands for $\dot{\boldsymbol{c}}_{\boldsymbol{t}} = \max\{\iota(\boldsymbol{c_{t_1}}, t - t_1), \boldsymbol{c_t}\}, \ddot{\boldsymbol{c}}_{\boldsymbol{t}} = \max\{\iota(\boldsymbol{c_{t_2}}, t - t_2), \boldsymbol{c_t}\}$ *where* $\boldsymbol{c_t}, \dot{\boldsymbol{c}}_{\boldsymbol{t}}, \ddot{\boldsymbol{c}}_{\boldsymbol{t}} \in \mathcal{C}(t)$.

Proof. The structure of the proof is similar to Proposition 7. Beyond the base cases, we first try every possible starting time t to schedule job j. Then we follow the scheduling policy that whenever a time slot is active, we try every possibility of $\boldsymbol{c_t}$. Depending on the time slot t where job j is scheduled, we divide the analysis into three cases.

Case 1) $t = t_2, q = m$. In this case, all the available machines are already reserved, therefore it is infeasible to assign job j to time slot t_2.

Case 2) $t = t_2, 0 < q < m$. In this case, job j is assigned to time slot t_2. Since $q > 0$, the calibration setting on time slot t_2 is already determined, we only need to choose an unoccupied machine for job j at slot t_2 and recurse to the sub-problem $f(j-1, t_1, t_2, q+1, \boldsymbol{c_{t_1}}, \boldsymbol{c_{t_2}})$.

Case 3) In this case, job j can be scheduled at time slot t, and time slot t is not yet active so far. We enumerate the calibrations that cover time slot t, i.e., vector $\boldsymbol{c_t} \in \mathcal{C}(t)$. As argued in Proposition 7, when job j is scheduled at time slot t, jobs $J(j-1, t_1, t)$ and $J(j-1, t, t_2)$ should be scheduled during intervals $(t_1, t]$ and $(t, t_2]$ respectively. Then We determine the calibrations that cross the boundary of the intervals. We define $\dot{\boldsymbol{c}}_{\boldsymbol{t}} = \max\{\iota(\boldsymbol{c_{t_1}}, t - t_1), \boldsymbol{c_t}\}$ to include all the calibrations in $\boldsymbol{c_{t_1}}$ or $\boldsymbol{c_t}$ and restrict $\dot{\boldsymbol{c}}_{\boldsymbol{t}} \in \mathcal{C}(t)$ to avoid the situation where the number of available machines exceeds m. We reserve one machine for the job j at time slot t in the first sub-problem $f(j-1, t_1, t, 1, \boldsymbol{c_{t_1}}, \dot{\boldsymbol{c}}_{\boldsymbol{t}})$. Symmetrically, we define vector $\ddot{\boldsymbol{c}}_{\boldsymbol{t}} = \max\{\iota(\boldsymbol{c_{t_2}}, t - t_2), \boldsymbol{c_t}\}$ and the second sub-problem to be $f(j-1, t, t_2, q, \ddot{\boldsymbol{c}}_{\boldsymbol{t}}, \boldsymbol{c_{t_2}})$. As we have tried every possibility of time slot t, vector $\boldsymbol{c_t}$, at least one try is the same as the optimal solution. ◀

Time Complexity. The dynamic program has table size $O(n^2|\Phi|^2|\mathcal{C}|^2)$. Constructing the solution from the sub-problems takes $O(|\Phi||\mathcal{C}|)$ steps. In total, the running time is $O(n^2|\Phi|^3|\mathcal{C}|^3) = O(n^8 m^{3T})$. When T is constant, our dynamic programming approach has polynomial running time.

Connection with Section 3. When m is input, the dynamic programming approach in Section 3 has exponential running time, while the running time of the approach in this section is polynomial on n and m. However, this running time is exponential on T. When T is constant, our proposed dynamic programming approach has polynomial running time.

5 PTAS

In this section, we extend the dynamic programming approach in Section 3 and present a PTAS. In other words, for any constant $\epsilon > 0$, we give a $(1+\epsilon)$ - approximation solution with m machines. The high level idea of the PTAS is to compress the vectors by decreasing the number of possible distinct starting times of calibrations so that the dynamic programming has polynomial running time. We propose a compression method by delaying the calibrations in the optimal solution and prove that in the approximation solution, for each time slot t the corresponding vector set $\Gamma(t)$ has polynomial cardinality. More specifically, we prove that given $(1+\epsilon)m$ machines there exists a $(1+\epsilon)$ - approximation solution such that for any time slot t, the number of distinct starting times (and ending times) of the calibrations that cover time slot t is at most $2\lceil 1/\epsilon\rceil + 1$. At last, we use a modified dynamic programming algorithm to find that $(1+\epsilon)$ - approximation solution without using the extra ϵm machines.

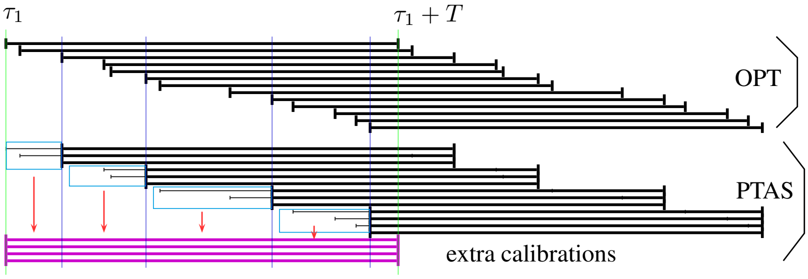


Figure 3 An illustration for Lemma 11 to show the transformation of the calibrations in the optimal solution. For each group of calibrations, after delaying the calibrations the affected jobs are depicted within a rectangle. We move the affected jobs in each rectangle into the extra machines, without changing the job starting time. As the time interval covered by each rectangle is disjoint with other rectangles, the new schedule of the jobs from two rectangles is independent of each other.

▶ **Lemma 11.** *There exists a $(1+\epsilon)$ - approximation solution on $(1+\epsilon)m$ machines such that for any time slot t, the number of distinct starting times (and ending times) of the calibrations that cover time slot t is at most $2\lceil 1/\epsilon\rceil + 1$, and there are at most m jobs scheduled at time slot t.*

Proof. Consider the optimal solution that satisfies Lemma 1 and Lemma 3, in which no two calibrations overlap with each other on the same machine. We show how to transform the optimal solution into another solution satisfying the statement, shown in Figure 3. In our approach, we maintain the schedule of the jobs as in the optimal schedule and only change the schedule of calibrations. Hence, in the new schedule there are at most m jobs scheduled at any time slot. Assume that in the optimal solution the calibrations are sorted in non-decreasing order of their starting times (regardless of the machines). We define *block* to be the set of consecutive calibrations satisfying the property that the largest difference of their starting times is less than T and the set is maximal in the sense that adding one more calibration will violate the property. We partition the calibrations in the optimal solution into many disjoint blocks starting from the first calibration. The transformation works in many phases where in each phase we work on one block.

Suppose in the current phase the block contains l calibrations. First, we will delay these l calibrations so that the number of distinct starting times of these calibrations is at most $\lceil 1/\epsilon\rceil$ (skip the phase and do nothing if $l \le \lceil 1/\epsilon\rceil$). This process will cause the infeasibility of some jobs because of the delay of calibrations. We construct a feasible schedule of jobs by creating an extra of $\lfloor \epsilon l\rfloor$ calibrations on the extra ϵm machines and rescheduling the affected jobs on the extra machines.

Let τ_i be the starting time of the i-th calibration in the block, we have $\tau_1 \le \tau_2 \le ... \le \tau_l < \tau_1 + T$. Note that $l \le m$ because each of these calibrations covers time slot $\tau_1 + T$.

Step 1. (Partition) We partition the calibrations in the block into $\lceil 1/\epsilon \rceil$ groups such that each group contains at most $\lceil \epsilon l \rceil$ consecutive calibrations. One would find that the partition is feasible as $\lceil 1/\epsilon \rceil \cdot \lceil \epsilon l \rceil \geq l$. Note that the maximum calibration starting time in one group is no larger than the minimum calibration starting time in the next group.

Step 2. (Delay) For each group of calibrations, let τ be the latest calibration starting time, then we delay the calibrations in this group so that they have identical starting time τ.

Step 3. (Augment) We add an extra group of $\lfloor \epsilon l \rfloor$ calibrations once with identical starting time τ_1 on the extra ϵm machines.

Step 4. (Transform) For the calibrations that are delayed in each group, we reschedule the corresponding affected jobs on the extra machines, without changing the starting time of any job.

Analysis. We show that the new schedule of jobs is feasible. First, we claim that the number of the affected jobs which start at a time $t \in [\tau_1, \tau_1 + T)$ in the optimal solution is at most $\lceil \epsilon l \rceil - 1$. In total we have created an extra of $\lfloor \epsilon l \rfloor$ calibrations. For each group of calibrations, let t_a, t_b be the smallest and largest calibration starting time, respectively. There is at least one calibration that is not delayed, hence we delay at most $\lceil \epsilon l \rceil - 1$ calibrations in this group. The affected jobs caused by the delay of the calibrations in this group must have job starting time within $[t_a, t_b)$. In other words, in the optimal solution any affected job that starts at a time within $[t_a, t_b)$ must be scheduled in a calibration from this group (i.e., not from other groups), because the maximum calibration starting time in one group is no larger than the minimum calibration starting time in the next group by Step 1. Therefore, $\forall t \in [\tau_1, \tau_1 + T)$, the total number of the affected jobs which start at time t is at most $\lceil \epsilon l \rceil - 1$, because we delay at most $\lceil \epsilon l \rceil - 1$ calibrations in this group. Since $\lceil \epsilon l \rceil - 1 \leq \lfloor \epsilon l \rfloor$, we conclude that the new schedule of jobs is feasible.

Now, we prove the lemma. Suppose in total there are b phases and let $\tau'_1, \tau'_2, ..., \tau'_b$ be the starting times of the extra calibrations in each phase. Then we have $\tau'_{i+1} - \tau'_i \geq T$, $\forall i \in [1, b)$ according to the above process. In one phase, we process l calibrations and we create $\lfloor \epsilon l \rfloor$ extra calibrations, which implies that the final solution is $(1+\epsilon)$-approximation since we never consider a calibration twice in different phases. Moreover Lemma 3 holds for the new solution because we do not change the starting time of any job and the starting time of the extra calibrations in each phase is the same as one of the calibrations from the block in the optimal solution. And note that in the new solution, calibrations might overlap with each other on the same machines. In each phase, we partition the calibrations into $\lceil 1/\epsilon \rceil$ groups and the calibrations in each group have identical starting time, hence the number of distinct starting times of the calibrations is at most $\lceil 1/\epsilon \rceil + 1$ in each phase, including the extra calibrations. In other words, for each interval $[\tau'_i, \tau'_{i+1})$ the number of distinct starting times of the calibrations that start during this interval in the new solution is at most $\lceil 1/\epsilon \rceil + 1$. Consider an arbitrary time slot t from $[\tau'_i, \tau'_{i+1})$ and the calibrations that cover slot t in the new solution. These calibrations must have starting times in $(\tau'_{i-1}, \tau'_{i+1})$ since $\tau'_{i+1} - \tau'_{i-1} \geq 2T$. Therefore, the total number of distinct starting times of the calibrations that cover time slot t is at most $2\lceil 1/\epsilon \rceil + 1$ (the extra calibrations in phase $i-1$ will not cover time slot t). Also, the total number of distinct ending times of the calibrations that cover time slot t is at most $2\lceil 1/\epsilon \rceil + 1$, because all calibrations have identical length. Eventually, the lemma is proved. ◀

Vector Compression

Lemma 11 shows that for each optimal solution, there is a corresponding $(1+\epsilon)$-approximation solution on $(m+\epsilon m)$ machines with the property that the number of distinct starting times of the calibrations that cover each time slot t is bounded by a constant. In the following, we propose a method to find a $(1+\epsilon)$-approximation solution with m machines, which is based on the dynamic programming in the previous section. We first propose the algorithm with $(m+\epsilon m)$ machines (which we refer to as *resource-augmentation* version), while guaranteeing that for any time slot there are at most m jobs scheduled. Note that the extra ϵm machines is due to the extra additional calibrations as shown in Figure 3. Therefore, we then transform the resource-augmentation solution to a solution that only requires m machines by rescheduling the calibration without changing the schedule of jobs.

Let $h = 2\lceil 1/\epsilon \rceil + 1$, $m' = m + \lfloor \epsilon m \rfloor$ and we assume $m' < n$ (a trivial solution could be found when the optimal schedule uses m' machines with $m' \geq n$). Previously in Section 3, for the calibrations that cover time slot t, we use vector $\boldsymbol{\gamma} = \langle \gamma_1, \gamma_2, ..., \gamma_m \rangle \in \Gamma(t)$ to mark the calibration starting time on each machine. Similar to Section 4, in the modified dynamic programming for PTAS, we discard the information of the mapping from calibrations to machines and only mark the starting times of the calibrations that cover time slot t. In other words, for each calibration, we do not need to know the corresponding machine on which the calibration takes effect. We focus on the solution on m' machines and use *configurations* to mark the calibration starting times.

▶ **Definition 12.** *We define a* configuration *to be a pair of vectors $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ where $\boldsymbol{\alpha} = \langle \alpha_1, \alpha_2, ..., \alpha_h \rangle$, $\boldsymbol{\eta} = \langle \eta_1, \eta_2, ..., \eta_h \rangle$ and for each $i \in [1,h]$, $\alpha_i \in \{\textsc{nul}\} \cup \Psi$ indicates the starting time of a calibration, $\eta_i \in [0, m']$ indicates the number of the calibrations that share the same starting time α_i. We define $\mathcal{A} = \{\langle \alpha_1, \alpha_2, ..., \alpha_h \rangle \mid \alpha_i \in \{\textsc{nul}\} \cup \Psi, \forall i \in [1,h]\}$ to be the set of all possible vectors $\boldsymbol{\alpha}$. Given time slot t, we define $\mathcal{A}(t) = \{\langle \alpha_1, \alpha_2, ..., \alpha_h \rangle \mid \alpha_i \in \{\textsc{nul}\} \cup (\Psi \cap [t-T, t)), \forall i \in [1,h]\}$, where α_i indicates the starting time of a calibration which covers time slot t. Let $\mathcal{B} = \{\boldsymbol{\eta} \mid \sum_{i=0}^{h} \eta_i \leq m', \eta_i \in [0, m'], \forall i \in [1,h]\}$ be the set of all possible vectors $\boldsymbol{\eta}$.*

In total we have at most $n^h|\Psi|^h$ configurations as $|\mathcal{A}| \leq |\Psi|^h$ and $|\mathcal{B}| \leq (m'+1)^h \leq n^h$, which implies that the total number of possible configurations is polynomial in n. Given time slot t, and two configurations $\langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle$ and $\langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle$, we use $\cup_t$ as the notation of the process that merges these two configurations into a new configuration $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ where $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle = \langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle \cup_t \langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle$ such that each calibration from the new configuration covers time slot t. In the merging process, we first identify the distinct starting times of the calibrations from the two configurations that cover time slot t, then for each distinct starting time, we count the number of calibrations that share the same starting time. We guarantee that $\boldsymbol{\alpha} \in \mathcal{A}(t)$ and $\boldsymbol{\eta} \in \mathcal{B}$ for the new configuration $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ after the merging process. In other words, we will discard the new configuration if it is not valid (either the number of distinct calibration starting times is beyond h or $\boldsymbol{\eta} \notin \mathcal{B}$).

▶ **Definition 13.** *We define $f^{\#}(j, t_1, t_2, q, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle)$ to be the minimum number of extra necessary calibrations to schedule jobs $J(j, t_1, t_2)$ on m' machines where $j \in J, t_1 \in \Phi$, $t_2 \in \Phi$, $q \in [0, m]$, $\check{\boldsymbol{\alpha}} \in \mathcal{A}(t_1), \hat{\boldsymbol{\alpha}} \in \mathcal{A}(t_2)$, $\check{\boldsymbol{\eta}} \in \mathcal{B}, \hat{\boldsymbol{\eta}} \in \mathcal{B}$ on the condition that*

- **i.)** *jobs in $J(j, t_1, t_2)$ are only scheduled during time interval $(t_1, t_2]$.*
- **ii.)** *calibrations indicated by configurations $\langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$ and $\langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$ have already been selected to be assigned to machines.*
- **iii.)** *q other jobs (not from $J(j, t_1, t_2)$) have already been assigned at time slot t_2.*
- **iv.)** *there are at most m jobs scheduled at any time slot.*

In dynamic programming, we allow the overlap of calibrations and we guarantee that at each time slot, we assign at most m jobs into this slot.

▶ **Proposition 14.** *Let $F^{\#} = f^{\#}(j, t_1, t_2, q, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle)$. For the base case $J(j, t_1, t_2) = \emptyset$, we set $F^{\#}$ to be 0 if time slot t_2 is covered by at least q calibrations given by configurations $\langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$ and $\langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$, otherwise ∞. If $j \notin J(j, t_1, t_2)$, we have $F^{\#} = f^{\#}(j - 1, t_1, t_2, q, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle)$. If $j \in J(j, t_1, t_2)$ and $\Phi(j) \cap (t_1, t_2] = \emptyset$, we have $F^{\#} = \infty$. Otherwise, we have $F^{\#} =$*

$$\min_{t \in \Phi(j) \cap (t_1, t_2]} \begin{cases} \infty & , \text{ if } t = t_2, q = m \\ f^{\#}(j-1, t_1, t_2, q+1, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle) & , \text{ if } t = t_2, 0 < q < m \\ \min\limits_{cond.} \quad \sum_{i=0}^{h} \eta_i & \\ + f^{\#}(j-1, t_1, t, 1, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle) & \\ + f^{\#}(j-1, t, t_2, q, \langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle) & , \text{ if } t < t_2 \text{ or } q = 0 \end{cases}$$

where cond. stands for $\langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle \cup_t \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$, $\langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle \cup_t \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$ where $\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\alpha}} \in \mathcal{A}(t)$ and $\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}, \ddot{\boldsymbol{\eta}} \in \mathcal{B}$.

Proof. Similar to Proposition 7, we maintain the invariant that whenever a time slot becomes active (i.e., we reserve a time slot for a job), we enumerate all the calibrations that cover this time slot (excluding the calibrations that have already been selected). For example, for time slot t which is not active at the moment, even some calibrations from configurations $\langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$, $\langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$ might already cover time slot t. We enumerate the remaining calibrations that could cover time slot t (i.e., configuration $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ with $\boldsymbol{\alpha} \in \mathcal{A}(t), \boldsymbol{\eta} \in \mathcal{B}$) when we plan to assign a job to time slot t. Note that the actual calibrations that cover time slot t come from configurations $\langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$, $\langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$ and $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$. In the dynamic programming, we enumerate the time slot t in which job $j \in J(j, t_1, t_2)$ is scheduled in the optimal schedule.

Case 1) If $t = t_2, q = m$, we cannot assign job j to time slot t because there are already other m jobs assigned to time slot t.

Case 2) $t = t_2, 0 < q < m$. In this case, time slot t has already become active since $q > 0$. Hence, the calibrations that cover time slot t_2 has already been enumerated (in other words, we do not need to enumerate it again). Also, job j could be assigned to time slot t_2 because $q < m$. Therefore, we just assign job j to time slot t and reduce to the sub-problem $f^{\#}(j - 1, t_1, t_2, q + 1, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle)$. Especially, if $q = 0$, no job has been assigned to time slot t, i.e., time slot t is not active, then we have to enumerate the calibrations that cover time slot t, which is handled in Case 3). If $t < t_2$, time slot t is not active because we only reserve time slot t_2 for the q other jobs.

Case 3) In this case, job j can be scheduled at time slot t, and no job has been assigned to time slot t so far. We enumerate the calibrations that cover time slot t, which is the configuration $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ with $\boldsymbol{\alpha} \in \mathcal{A}(t), \boldsymbol{\eta} \in \mathcal{B}$. As argued in Proposition 7, when job j is scheduled at time slot t, jobs $J(j - 1, t_1, t)$ and $J(j - 1, t, t_2)$ should be scheduled during intervals $(t_1, t]$ and $(t, t_2]$ respectively. Hence we reduce to sub-problems $f^{\#}(j - 1, t_1, t, 1, \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle, \langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle)$ and $f^{\#}(j - 1, t, t_2, q, \langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle, \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle)$ where $\langle \dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\eta}} \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle \cup_t \langle \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\eta}} \rangle$, $\langle \ddot{\boldsymbol{\alpha}}, \ddot{\boldsymbol{\eta}} \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle \cup_t \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\eta}} \rangle$ by reserving a calibrated machine at time slot t for job j. Because we enumerate all possible configurations $\langle \boldsymbol{\alpha}, \boldsymbol{\eta} \rangle$ with $\boldsymbol{\alpha} \in \mathcal{A}(t), \boldsymbol{\eta} \in \mathcal{B}$, we are able to reach the schedule that is the same as the optimal schedule. ◀

Time complexity. The modified dynamic program has table size $O(n^2 |\Phi|^2 (|\mathcal{A}||\mathcal{B}|)^2)$. Constructing the solution from the sub-problems takes $O(|\Phi||\mathcal{A}||\mathcal{B}|)$ steps. In total the time complexity is $O(n^2 |\Phi|^3 (|\mathcal{A}||\mathcal{B}|)^3) = O(n^2 |\Phi|^3 n^{3h} |\Psi|^{3h}) = O(n^{17+18\lceil 1/\epsilon \rceil})$.

▶ **Lemma 15.** *There exists a $(1+\epsilon)$ - approximation solution on m machines.*

Proof. By Lemma 11 and the algorithm in Proposition 14, we can obtain a $(1+\epsilon)$ - approximation solution σ on m' machines while guaranteeing that there are at most m jobs scheduled at any time slot. In the following, we transform σ to an $(1+\epsilon)$ - approximation solution σ' on m machines, without changing the schedule of jobs. In solution σ, we consider the earliest time slot t such that more than m machines are calibrated at time slot t.

Case 1) If no such time slot t exists, we then assign the calibrations to m machines via Round-Robin method, as proposed in [3], and obtain a feasible solution on m machines.

Case 2) Otherwise, there must be some calibration starting at time $t-1$, we then delay this calibration by one more time slot. Note that jobs are still feasible since there are at most m jobs scheduled at time slot t. We repeat the above process until Case 1) occurs and Case 1) eventually will occur since the sum of the calibration starting time is increasing after each delay operation. Thus, we finish the proof. ◀

Connection with Section 4. Note that in Section 3 we directly record the calibration times on each machine in the dynamic program, while in Section 4 and this section we propose different methods for vector compression to ensure a polynomial space of the proposed dynamic programs. Specifically, in Section 4, for a fixed time slot t, the number of possible distinct calibration starting times within interval $[t, t+T)$ is constant since T is constant, instead, in this section when T is input, we apply the calibration delaying operation to ensure the polynomial number of distinct calibration starting times within interval $[t, t+T)$.

6 Conclusion

We study the scheduling problem with calibrations on multiple machines where we consider the schedule of unit-time processing jobs with release times and deadlines such that the total number of calibrations is minimized. We propose two dynamic programming approaches to solve the problem with running time $O(n^{8+6m})$ and $O(n^8 m^{3T})$ respectively. Thus when m is constant or T is constant, we can give an algorithm of polynomial running time. Moreover, we present a PTAS, which has running time $O(n^{17+18\lceil 1/\epsilon \rceil})$. This approach very likely works only on the case that the jobs have identical processing time. It would be worth challenging to tackle the open problem proposed by Bender et al. [3] about the complexity status on multiple machines with jobs of unit processing times.

References

1 Eric Angel, Evripidis Bampis, Vincent Chau, and Vassilis Zissimopoulos. On the complexity of minimizing the total calibration cost. In *International Workshop on Frontiers in Algorithmics*, pages 1–12. Springer, 2017. doi:10.1007/978-3-319-59605-1_1.

2 Eric Angel, Evripidis Bampis, Vincent Chau, and Vassilis Zissimopoulos. Calibrations scheduling with arbitrary lengths and activation length. *Journal of Scheduling*, 24(5):459–467, 2021. doi:10.1007/S10951-021-00688-5.

3 Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. Efficient scheduling to minimize calibrations. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 280–287, New York, NY, USA, 2013. ACM. doi:10.1145/2486159.2486193.

4 B Bringmann, A Küng, and W Knapp. A measuring artefact for true 3d machine testing and calibration. *CIRP Annals-Manufacturing Technology*, 54(1):471–474, 2005.

5 Peter Brucker and P Brucker. *Scheduling algorithms*, volume 3. Springer, Heidelberg, 2007.

6 Chris Burroughs. New integrated stockpile evalution program to better ensure weapons stockpile safety, security, reliability, 2006. URL: http://www.sandia.gov/LabNews/060331.html.

7 Vincent Chau, Shengzhong Feng, Minming Li, Yinling Wang, Guochuan Zhang, and Yong Zhang. Weighted throughput maximization with calibrations. In *Algorithms and Data Structures: 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5–7, 2019, Proceedings 16*, pages 311–324. Springer, 2019. doi:10.1007/978-3-030-24766-9_23.

8 Vincent Chau, Minming Li, Samuel McCauley, and Kai Wang. Minimizing total weighted flow time with calibrations. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, pages 67–76, New York, NY, USA, 2017. ACM. doi:10.1145/3087556.3087573.

9 Vincent Chau, Minming Li, Elaine Yinling Wang, Ruilong Zhang, and Yingchao Zhao. Minimizing the cost of batch calibrations. *Theoretical Computer Science*, 828:55–64, 2020. doi:10.1016/J.TCS.2020.04.020.

10 Hua Chen, Vincent Chau, Lin Chen, and Guochuan Zhang. Scheduling many types of calibrations. In *International Conference on Algorithmic Applications in Management*, pages 286–297. Springer, 2020. doi:10.1007/978-3-030-57602-8_26.

11 Zuzhi Chen and Jialin Zhang. Online scheduling of time-critical tasks to minimize the number of calibrations. *Theoretical Computer Science*, 914:1–13, 2022. doi:10.1016/J.TCS.2022.01.040.

12 Jeremy T. Fineman and Brendan Sheridan. Scheduling non-unit jobs to minimize calibrations. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 161–170, New York, NY, USA, 2015. ACM. doi:10.1145/2755573.2755605.

13 Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

14 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, October 1975. doi:10.1145/321906.321909.

15 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, July 2000. doi:10.1145/347476.347479.

16 Kuo-Huang Lin and Bin-Da Liu. A gray system modeling approach to the prediction of calibration intervals. *IEEE Transactions on Instrumentation and Measurement*, 54(1):297–304, 2005. doi:10.1109/TIM.2004.840234.

17 Hoai-Nhan Nguyen, Jian Zhou, and Hee-Jun Kang. A new full pose measurement method for robot calibration. *Sensors*, 13(7):9132–9147, 2013. doi:10.3390/S130709132.

18 Emilia Nunzi, Gianna Panfilo, Patrizia Tavella, Paolo Carbone, and Dario Petri. Stochastic and reactive methods for the determination of optimal calibration intervals. *IEEE Transactions on Instrumentation and Measurement*, 54(4):1565–1569, 2005. doi:10.1109/TIM.2005.851501.

19 Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th. ACM Symposium on Theory of Computing (STOC '97)*, pages 140–149, New York, NY, 1997. ACM Press.

20 SR Postlethwaite, DG Ford, and D Morton. Dynamic calibration of CNC machine tools. *International Journal of Machine Tools and Manufacture*, 37(3):287–294, 1997.

21 Sartaj K. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, January 1976. doi:10.1145/321921.321934.

22 Petra Schuurman and Gerhard J. Woeginger. Approximation schemes – a tutorial, 2007.

23 Minze Stuiver, Paula J Reimer, and Thomas F Braziunas. High-precision radiocarbon age calibration for terrestrial and marine samples. *Radiocardbon*, 40(3):1127–1151, 1998.

24 Kai Wang. Calibration scheduling with time slot cost. *Theoretical Computer Science*, 821:1–14, 2020. doi:10.1016/J.TCS.2020.03.018.

25 Gerhard J Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000. doi:10.1287/IJOC.12.1.57.11901.

26 G Zhang and R Hocken. Improving the accuracy of angle measurement in machine calibration. *CIRP Annals-Manufacturing Technology*, 35(1):369–372, 1986.

27 Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. doi:10.1109/34.888718.

Mimicking Networks for Constrained Multicuts in Hypergraphs

Kyungjin Cho
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Eunjin Oh
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Abstract

In this paper, we study a *multicut-mimicking network* for a hypergraph over terminals T with a parameter c. It is a hypergraph preserving the minimum multicut values of any set of pairs over T where the value is at most c. This is a new variant of the multicut-mimicking network of a graph in [Wahlström ICALP'20], which introduces a parameter c and extends it to handle hypergraphs. Additionally, it is a natural extension of the *connectivity-c mimicking network* introduced by [Chalermsook et al. SODA'21] and [Jiang et al. ESA'22] that is a (hyper)graph preserving the minimum cut values between two subsets of terminals where the value is at most c.

We propose an algorithm for a hypergraph that returns a multicut-mimicking network over terminals T with a parameter c having $|T|c^{O(r\log c)}$ hyperedges in $p^{1+o(1)}+|T|(c^r\log n)^{\tilde{O}(rc)}m$ time, where p and r are the total size and the rank, respectively, of the hypergraph.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases hyperedge multicut, vertex sparsification, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.21

Related Version *Full Version*: https://arxiv.org/abs/2409.12548

Funding This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00358505).
Kyungjin Cho: Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00410835).

1 Introduction

Graph sparsification is a fundamental tool in theoretical computer science. By reducing the size of a graph while preserving specific properties, such as the value of an objective function or its approximation, graph sparsification significantly enhances computational efficiency. This is particularly crucial for practical applications with limited resources and for handling large-scale data in real-world problems. Due to these advantages, various types of sparsification results have been presented over the decades, including spanners [4, 9], flow sparsification [6, 15], and cut sparsification [5]. Additionally, their applications have been widely studied, such as in designing dynamic algorithms [11]. In this paper, we focus on graph sparsification specifically tailored for hypergraph separation and cut problems.

Hypergraph separation and cut problems have garnered significant attention due to their extensive applications and theoretical challenges. These problems are particularly compelling because hypergraphs offer more accurate modeling of many complex real-world scenarios compared to normal graphs. Examples include VLSI layout [1], data-pattern-based clustering [23], and social tagging networks [25]. The transition from graph to hypergraph separation problems opens up new avenues for research, driven by the need

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 21; pp. 21:1–21:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to address the unique properties and complexities inherent in hypergraphs. Researchers have thus increasingly focused on developing graph algorithms and theoretical frameworks for hypergraph problems, such as the small set expansion problem in hypergraphs [19], spectral sparsification in hypergraphs [2, 13], and connectivity-c mimicking problem in hypergraphs [10]. This growing interest underscores the critical importance of hypergraph separation and cut problems in both theoretical and practical applications.

One of the key problems in (hyper)graph sparsification is the *mimicking problem*. It aims to find a graph that preserves minimum cut sizes between any two subsets of vertices called *terminals*. A *cut* between two sets of vertices is a set of edges whose removal disconnects the given two sets. Kratsch et al. [14] showed that there is a mimicking network with $O(\tau^3)$ edges, where τ is the number of edges incident to terminals. Chalermsook et al. [3] introduced a *constraint version*, called *connectivity-c mimicking problem*, that aims to preserve minimum cut sizes between every two subsets of terminals where the size is at most c, and they showed that there is such a graph with $O(kc^4)$ edges, which was later improved to $O(kc^3)$ [16], where k is the number of terminals. This result was extended to hypergraphs by Jiang et al. [10].

A crucial variant of the mimicking problem is the *multicut-mimicking problem*. A *multicut* of pairs of vertices is a set of (hyper)edges whose removal disconnects each given pair. Studies have shown that the multicut problem is highly beneficial in various applications, including network design, optimization, and security, where maintaining specific connectivity while minimizing resources is necessary compared to cut problems [12]. It is already known that the problem is NP-hard even for graphs [8, 20]. A *multicut-mimicking network* for a set of terminals in a (hyper)graph is a (hyper)graph that preserves the size of the minimum multicut for any set of pairs of terminals. Kratsch et al. [14] proposed a method to obtain a multicut-mimicking network by contracting edges in a graph except at most $\tau^{O(k)}$ edges, where k and τ are the numbers of terminals and incident edges to terminals, respectively. Wahlström [24] refined this method and reduced the number of edges to $\tau^{O(\log \tau)}$.

Unlike the mimicking problem, there is no existing result for the constraint version of *multicut-mimicking network problem*, even for graphs. We further study the multicut-mimicking problem by introducing a parameter c. Precisely, we present an algorithm to compute a hypergraph, that preserves the size of the minimum multicut for any set of pairs of terminals where the size is at most c, with a linear size in the number of terminals while the previous best-known result for multicut-mimicking network, without the parameter c, has an exponential size [14]. It will allow for more refined control over the sparsification process, enabling the construction of smaller and more efficient networks. For instance, this notion in mimicking problem was utilized for a dynamic connectivity problem [11].

Our result. In this paper, we study *vertex sparsifiers for multiway connectivity* with a parameter $c > 0$. Our instance (G, T, c) consists of an undirected hypergraph G, terminal set $T \subseteq V(G)$, and a parameter c. Precisely, we construct a hypergraph that preserves minimum multicut values over T where the value is at most c. It is the first result for the multicut-mimicking networks adapting the parameter c even for graphs.

Previously, the best-known multicut-mimicking network had a quasipolynomial size in the total degree of terminals in T [24], specifically $|\partial T|^{O(\log |\partial T|)}$. By introducing the parameter c, we demonstrate that a multicut-mimicking network for (G, T, c) exists with a size linear in $|T|$. This allows us to utilize the near-linear time framework of Jiang et al. [10] to find a mimicking network using the *expander decomposition* of Long and Saranurak [18]. Our result is summarized in Theorem 1. Here, $m = |E(G)|$ and r is the rank of G.

▶ **Theorem 1.** *For (G, T, c), we can compute a multicut-mimicking network of at most $kc^{O(r \log c)}$ hyperedges in $p^{1+o(1)} + k(c^{r \log c} \log n)^{O(rc)} m$ time, where $k = |T|$ and $p = \sum_{e \in E} |e|$.*

Outline. Our work extends the framework from connectivity-c mimicking networks for hypergraphs, introduced by Jiang et al. [10], to multicut-mimicking networks, as well as adapting methods from multicut-mimicking networks for graphs, introduced by Wahlström [24], to hypergraphs with a parameter c. While we broadly follow the previous approaches, we extend the concepts and methods used in the previous studies to fit the multicut-mimicking problem in hypergraphs with the parameter c. This extension allows us to handle the complexities of hypergraphs effectively.

We introduce notions used in this paper in Section 2 and illustrate an efficient algorithm to compute a small-sized multicut-mimicking network outlined in Theorem 1 in Section 3. We give an upper bound for the size of minimal multicut-mimicking networks of hypergraphs in Section 4, which is a witness for the performances of our algorithm outlined in Theorem 1.

2 Preliminaries

A *hypergraph* G is a pair $(V(G), E(G))$, where $V(G)$ denotes the set of vertices and $E(G)$ is a collection of subsets of $V(G)$ referred to as *hyperedges*. If the context is clear, we write V and E. The *rank* of G is defined as the size of its largest hyperedge. For a vertex $v \in V$, a hyperedge e is said to be *incident* to v if $v \in e$. For a vertex set $X \subset V$, let $\partial_G X$ denote the set of hyperedges in E containing at least one vertex from X and one from $V \setminus X$, and let $E(X)$ denote the set of hyperedges fully contained in X. Additionally, we let G/e denote the *contraction* of a hyperedge e in G obtained by merging all vertices in e into a single vertex and modifying the other hyperedges accordingly. A *path* in a hypergraph is defined as a sequence of hyperedges such that any two consecutive hyperedges contain a common vertex.

Consider a *partition* $(X_1, \ldots, X_s)$ of a vertex set $X \subseteq V$. We call each subset X_i a *component* of this partition. Additionally, the *cut* of $(X_1, \ldots, X_s)$ in G is defined as the set of hyperedges of G intersecting two different components. We let $[a] = \{1, \ldots, a\}$ and $[a, b] = \{a, \ldots, b\}$ for integers $a < b$. Furthermore, let $|X|$ denote the number of elements of a set X. For a hyperedge $e \in E(G)$, we let $|e|$ to denote the number of vertices in e.

In this paper, an instance (G, T, c) consists of a hypergraph G, a set $T \subseteq V(G)$, and a positive constant c. We refer to the vertices in T as *terminals*. For a set R of pairs of T, a *multicut* of R in G is a set of hyperedges $F \subset E(G)$ such that every connected component in $G \setminus F$ contains at most one element of every pair $\{t, t'\} \in R$. We construct a *multicut-mimicking network* H of (G, T, c) that is a hypergraph obtained from G by contraction of hyperedges which preserves the size of minimum multicut for all set R of pairs over T where the size is at most c. Precisely, if a multicut F of R exists in G with $|F| \leq c$, then a multicut F' of R exists in H with $|F'| \leq |F|$. We say H is *minimal* if no contraction H/e is a multicut-mimicking network of (G, T, c). Analogously, we define a *minimal instance.* We address the multicut-mimicking problem by utilizing *multiway cuts.*

2.1 Multiway Cuts and Essential Edges

We refer to a partition of terminals T as a *terminal partition.* For a terminal partition $\mathcal{T}$, a hyperedge set F is termed *a multiway cut of* $\mathcal{T}$ if any two terminals from different components in $\mathcal{T}$ are not connected in $G \setminus F$. Furthermore, if there is no multiway cut of size less than $|F|$, then F is called a *minimum multiway cut of* $\mathcal{T}$ *in* G. Let $\mathsf{min\text{-}cut}_G(\mathcal{T})$

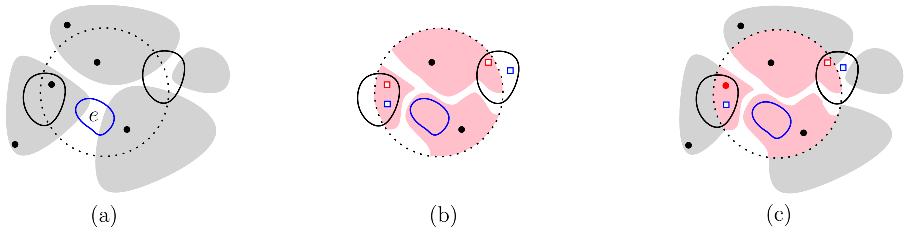


Figure 1 Sketch of Lemma 3. For a multiway cut F of a terminal partition $\mathcal{T}$ and $X \subset V$, illustrated in (a), let $\mathcal{T}'$ be the terminal partition of T_X in $(\hat{G}[X], T_X, c_X)$ according to $G \setminus F$. Then we can modify F as excluding e if e is non-essential in $(\hat{G}[X], T_X, c_X)$, illustrated in (b-c).

denote the minimum multiway cut size of the partition in G. A hyperedge $e \in E(G)$ is said to be *essential* for (G, T, c) if there exists a terminal partition $\mathcal{T}$ with $\textsf{min-cut}_G(\mathcal{T}) \leq c$ such that every minimum multiway cut of $\mathcal{T}$ in G contains e. Otherwise, e is *non-essential*.

Multicuts and multiway cuts in graphs are closely related [24, Proposition 2.2]. We observe that this close relation also holds in hypergraphs. Briefly, a contraction of a non-essential hyperedge is a multicut-mimicking network by Lemma 2, proved in the full version.

▶ **Lemma 2.** *For a hyperedge e of G, G/e is a multicut-mimicking network for (G, T, c) if and only if e is non-essential for (G, T, c).*

A multicut-mimicking network is minimal if and only if every hyperedge is essential. Note that we cannot contract multiple non-essential hyperedges simultaneously. This is because even if two hyperedges e and e' are non-essential in (G, T, c), the contraction $G/\{e, e'\}$ might not be a multicut-mimicking network of (G, T, c) even for a graph G, not a hypergraph. In this paper, we construct a multicut-mimicking network by finding and contracting non-essential hyperedges one by one.

2.2 Restricted Hypergraphs and Subinstances

The *subinstance* $(\hat{G}[X], T_X, c_X)$ of (G, T, c) for $X \subset V(G)$ is constructed as follows. Refer to Figure 1 (a-b). For each hyperedge $e \in \partial X$, we insert a vertex a_e, and we choose an arbitrary terminal t_e in $e \cap (X \cap T)$. If no such terminal exists, we insert a new vertex t_e. We refer to a_e, t_e as the *anchored terminals* of e, and $(e \cap X) \cup \{a_e, t_e\}$ as the *restricted hyperedge* of e, denoted by e_X. We obtain $\hat{G}[X]$ from G through the following: i) add the anchored terminals of the hyperedges in ∂X, ii) replace the hyperedges in ∂X with their restricted hyperedges, and iii) delete the vertices $V \setminus X$ and the hyperedges $E(V \setminus X)$. We call it the *restricted hypergraph* of G for X. Let T_X denote the set of all terminals in $T \cap X$ and the anchored terminals, and $c_X = \min\{c, |T_X|\}$. All subinstances preserve all essential hyperedges in the original instance by Lemma 3. Figure 1 sketches its proof, and details are in the full version.

▶ **Lemma 3.** *If a hyperedge e is non-essential in a subinstance $(\hat{G}[X], T_X, c_X)$, then e is also non-essential in the original instance (G, T, c). Furthermore, e is not in $\partial_G X$.*

3 Efficient Algorithm for Computing Multicut-Mimicking Networks

In this section, we design an algorithm to compute a minimal multicut-mimicking network for (G, T, c), where G is a hypergraph. We broadly follow the approach of Jiang et al. [10]. Since their original algorithm was designed for mimicking networks, not multicut-mimicking networks, we need to modify their algorithm. First, we introduce their algorithm briefly.

Jiang et al. [10] designed an algorithm to find a connectivity-c mimicking network for hypergraphs using the *expander decomposition* of Long and Saranurak [18]. Precisely, they designed an algorithm to find a connectivity-c mimicking network of size linear in $|T|$ for an *expander* G with terminals T, and then, they extended it for a general hypergraph using the expander decomposition. For a parameter $\phi > 0$, a hypergraph G (and instance (G, T, c)) is called a *ϕ-expander* if either $E(X)$ or $E(V \setminus X)$ has at most $\phi^{-1}|\partial X|$ hyperedges for any vertex set $X \subset V(G)$. The following explains the key idea of their and our algorithm.

Recall that contracting a non-essential hyperedge obtains a smaller mimicking network by Lemma 2. Generally, a non-essential hyperedge can be found by comparing every terminal partition and subset of hyperedges, which is time-consuming. However, if we suppose that the given instance is an *expander*, then we can do this more efficiently by comparing *useful terminal partitions* and their minimum multiway cuts instead of whole partitions and hyperedge subsets. We adapt concepts used in the previous research such as *useful terminal partitions* to suit our needs, and we newly introduce the concept *core* of a multiway cut which refers to a small-sized vertex set including whole hyperedges of the multiway cut.

Useful terminal partitions, connected multiway cuts, and cores. Assume that the instance (G, T, c) is a ϕ-expander and G is a connected hypergraph. For a multiway cut F in G, we define the *core* of F as the union C of connected components X in $G \setminus F$ with $|E(X)| \leq \phi^{-1}|F|$. The definition of an expander guarantees that at most one component in $G \setminus F$ has more than $\phi^{-1}|F|$ hyperedges. Therefore, the multiway cut F includes all hyperedges ∂C and is contained in $E(C) \cup \partial C$. We say F is a *connected multiway cut* in (G, T, c) if it is a minimum multiway cut of some terminal partition with $|F| \leq c$ and $T \cap C$ is connected in $\hat{G}[C]$, where C is the core of F and $\hat{G}[C]$ is the restricted hypergraph defined in Section 2.2. A terminal partition is said *useful* if every minimum multiway cut of it is a connected multiway cut.

Since every core of connected multiway cuts has a small number of vertices and hyperedges in ϕ-expander, we can enumerate all of them efficiently. Additionally, since a core includes its corresponding multiway cut, we can also enumerate all connected multiway cuts and useful partitions. Details are in Section 3.1. The most interesting property is that comparing all useful partitions is sufficient to find a non-essential hyperedge.

▶ **Lemma 4.** *A hyperedge $e \in E$ is essential for (G, T, c) if and only if there is a useful partition such that every minimum multiway cut for it contains e.*

Proof. The "if" direction is trivial since it is consistent with the definition of essential. For the "only if" direction, we assume that e is essential for (G, T, c). Let $\mathcal{T}$ be a terminal partition minimizing $\textsf{min-cut}_G(\mathcal{T})$ of which every minimum multiway cut involves e. In the following, we show that $\mathcal{T}$ is a *useful partition* by contradiction. Figure 2 illustrates this proof.

Assume that $\mathcal{T}$ is not a useful partition for (G, T, c), and let F be a minimum multiway cut of $\mathcal{T}$ which is not a connected multiway cut. If the core of F is $V(G)$, then F is a connected multiway cut. Therefore, the core is the complement of some connected component X in $G \setminus F$. Let C be the connected component in $G - X$ that intersects the hyperedge e. Refer to Figure 2(a). Then we decompose the multiway cut F into F_e and $\bar{F}$ where $F_e = F \cap E(C \cup X)$ and $\bar{F} = F \setminus F_e$. By the construction, we have $F_e \subsetneq F$ and $e \in F_e$. We construct a minimum multiway cut of $\mathcal{T}$ excluding e that completes the proof.

Let $\mathcal{T}'$ be the terminal partition according to $G \setminus F_e$. Since $F_e \subsetneq F$ and we chose $\mathcal{T}$ as minimizing $\textsf{min-cut}_G(\mathcal{T})$ while any minimum multiway cut of $\mathcal{T}$ contains e, there is a multiway cut F' of $\mathcal{T}'$ excluding e. We claim that $F' \cup \bar{F}$ is a minimum multiway cut of $\mathcal{T}$

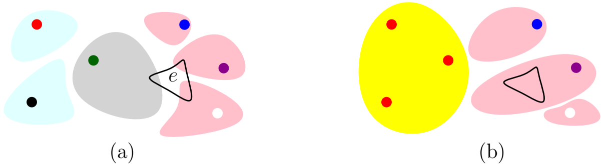


Figure 2 Illustration of the proof of Lemma 4. (a) Illustration of the terminal partition $\mathcal{T}$ and the vertex partition according to $G \setminus F$. The middle gray area is X. The right three red areas form C. (b) Illustration of the terminals partition $\mathcal{T}'$ and the partition of $G \setminus F'$. (c) Illustration of the partition $G \setminus (F' \cup \bar{F})$. $F' \cup \bar{F}$ is a multiway cut of $\mathcal{T}$ excluding e.

excluding e which contradicts and completes the proof. Refer to Figure 2(b-c). Note that the multiway cut has a size at most $|F|$ and excludes e by construction. Thus, it is sufficient to show that there is no path in $G \setminus (F' \cup \bar{F})$ between two components in $\mathcal{T}$.

Consider a path π in G between two terminals in different components in $\mathcal{T}$. Note that π is not a path in $G \setminus F$, and thus, π is not in $G \setminus F_e$ or $G \setminus \bar{F}$. Recall that F' is the multiway cut of the terminal partition according to $G \setminus F_e$. That means π is not in $G \setminus F'$ if it is not in $G \setminus F_e$. Therefore, there is no path in $G \setminus (F' \cup \bar{F})$ between two components in $\mathcal{T}$. ◀

3.1 Useful Terminal Partitions in Expanders

In this section, we explain how to efficiently enumerate all useful terminal partitions and their minimum multiway cuts. Then, we explain how to compute a multicut-mimicking network for an expander using the enumerated list along with Lemma 4. Here, the instance (G, T, c) is a ϕ-expander and G is a connected hypergraph.

The key is based on Observation 5, proved in the full version. Briefly, cores in a ϕ-expander have a small number of vertices and hyperedges. The observation enables us to find all cores of connected multiway cuts and subsequently enumerate all useful partitions. However, it is possible to enumerate terminal partitions that are not useful. Therefore, we need to *prune* the enumerated lists for useful terminal partitions and their minimum multiway cuts.

▶ **Observation 5.** *For a connected multiway cut F and its core C, the restricted hypergraph $\hat{G}[C]$ is connected and has at most $(3\phi^{-1} + 1)|F|$ hyperedges.*

Enumerating connected multiway cuts. Jiang et al. [10, EnumerateCutsHelp] designed an algorithm to enumerate all connected vertex sets C with $|\partial C| \leq c$, $|E(C)| \leq M$, and $t \in C$ in the ϕ-expander G when a vertex $t \in V(G)$ and two integers c, M are given as an input. This algorithm takes $(r(M + c))^{O(rc)}$ time. Additionally, the number of returned connected vertex sets is at most $(r(M + c))^{O(rc)}$. We use this algorithm along with Observation 5 for enumerating all connected multiway cuts. Details are in the full version. Briefly, we find all connected vertex set C with $C \cap T \neq \emptyset$, $|\partial C| \leq c$, and $|E(C)| \leq (3\phi^{-1} + 1)c$. Then we can enumerate every connected multiway cut from c sized subsets of $E(C) \cup \partial C$ by Observation 5.

In conclusion, we enumerate $|T|(rc\phi^{-1})^{O(rc)}$ multiway cuts including all connected multiway cuts of size at most c in $|T|(rc\phi^{-1})^{O(rc)}$ time. At most $|T|(rc\phi^{-1})^{O(rc)}$ terminal partitions of T are contributed by the enumerated multiway cuts since each multiway cut of size c generates at most rc connected components. They include all useful partitions.

Pruning useful terminal partitions. To check whether a terminal partition is useful or not, we need to verify if $C \cap T$ is connected in $\hat{G}[C]$ for each core C of its minimum multiway cuts. Specifically, it is sufficient to check the inclusion-wise minimal cores among them. For this, we utilize *important cuts* ∂R, which inclusion-wise maximizes $R \subset V$ while maintaining the size of the cut ∂R. The definition aligns with our needs as outlined in Lemma 6, proved in the full version.

For two disjoint vertex sets A and B, let R be a vertex set containing A while excluding B. We say ∂R is an *important cut* of (A, B) if there is no $R' \supsetneq R$ excluding B with $|\partial R'| \leq |\partial R|$. This definition holds even if $A = \emptyset$. In a directed graph, an important cut is defined analogously by setting ∂R as the outgoing arcs from R to $V \setminus R$. Furthermore, there is an FPT algorithm for enumerating all important cuts in a directed graph [7].

▶ **Lemma 6.** *For a terminal partition $\mathcal{T}$ with $\mathsf{min\text{-}cut}_G(\mathcal{T}) \leq c$, the following are equivalent:*
(i) *$\mathcal{T}$ is a useful terminal partition of T, and*
(ii) *Every minimum multiway cut F of $\mathcal{T}$ is a connected multiway cut if some component T' in $\mathcal{T}$ and important cut R of $(T', T \setminus T')$ satisfy $\partial R \subset F$, $|E(R)| \geq c\phi^{-1}$, and $F \cap E(R) = \emptyset$.*

We construct an auxiliary directed graph D^{inc} to enumerate important cuts in G. For instance (G, T, c), the vertex set of D^{inc} is the union of $V(G)$ and two copies E_{in} and E_{out} of $E(G)$. For a hyperedge e in $E(G)$, we use e_{in} and e_{out} to denote the copies of e in E_{in} and E_{out}, respectively, and we insert one arc from e_{in} to e_{out}. For each $v \in V(G)$ and $e \in E(G)$ with $v \in e$, we insert $2c$ parallel arcs from v to e_{in} and from e_{out} to v. Important cuts in G correspond one-to-one with those in D^{inc}. Details are in the full version.

We can enumerate all important cuts of G by applying the FPT algorithm for D^{inc}. There are at most $2^{O(rc)}$ number of important cuts in G, and they can be enumerated in $2^{O(rc)}m$ time [7], where $m = |E(G)|$. By Lemma 6, we can prune all useful terminal partitions in $|T|(rc\phi^{-1})^{O(rc)}m$ time among the $|T|(rc\phi^{-1})^{O(rc)}$ enumerated candidates.

Lemma 7 summarizes this section, details are in the full version. In the remainder, we give an algorithm to compute a minimal multicut-mimicking network for ϕ-expander using it.

▶ **Lemma 7.** *There are $|T|(rc\phi^{-1})^{O(rc)}$ useful partitions and their minimum multiway cuts in a ϕ-expander (G, T, c). We can enumerate all of them in $|T|(rc\phi^{-1})^{O(rc)}m$ time.*

Algorithm for ϕ-expanders. By Lemma 4 and Lemma 7, we can recursively find and contract a non-essential hyperedge efficiently for a ϕ-expander (G, T, c) until every hyperedge in the instance is essential. The algorithm is outlined in the following lemma. Recall that a *minimal multicut-mimicking network* H of (G, T, c) is a multicut-mimicking network so that every hyperedge in (H, T, c) is essential. Here, $m = |E(G)|$ and r is the rank of G.

▶ **Lemma 8.** *For a ϕ-expander (G, T, c), we can find a minimal multicut-mimicking network in $|T|(rc\phi^{-1})^{O(rc)}m$ time. Moreover, it has at most $|T|c^{O(r \log c)}$ hyperedges.*

Sketch of the proof. We demonstrate that a minimal multicut-mimicking network has at most $|T|c^{O(r \log c)}$ hyperedges in Section 4 which is one of our main contributions. We sketch an algorithm to compute a minimal multicut-mimicking network, details are in the full version.

If $m \in O(c\phi^{-1})$, then we can obtain such a multicut-mimicking network by enumerating all multiway cuts of size at most c. In the following, we consider the other case that $m \geq 3c\phi^{-1} + c$. At the beginning of the algorithm, we enumerate all important cuts and their minimum multiway cuts by applying Lemma 7. Since there are at most $k(rc\phi^{-1})^{O(rc)}$ multiway cuts and each multiway cut consists at most $(rc)^{O(rc)}$ useful partitions, we visit a hyperedge and check whether it is non-essential in the current instance in $k(rc\phi^{-1})^{O(rc)}$ time by Lemma 4. If it is non-essential, we contract it before we move to another hyperedge.

After we contract a non-essential hyperedge, we do not need to call the algorithm outlined in Lemma 7 again. Precisely, deleting multiway cuts which include the contracted hyperedge among the already enumerated ones is sufficient while more than $(3c\phi^{-1} + c)$ hyperedges are left. This is because no new connected multiway cut occurs or disappears by contracting a non-essential hyperedge if the number of remaining hyperedges exceeds $(3c\phi^{-1} + c)$. Its detailed proof is in the full version. If all remaining hyperedges are essential, then we return the current instance as a solution. For the other case that the number of them is at most $(3c\phi^{-1} + c)$, we apply the algorithm for $m \in O(c\phi^{-1})$ explained before. In conclusion, we can obtain a minimal multicut-mimicking network in the time complexity in Lemma 8. ◀

3.2 Near-Linear Time Algorithm for General Hypergraphs

In this section, we obtain a multicut-mimicking network for a general instance (G, T, c), by recursively calling MimickingExpander. The submodule MimickingExpander computes a small multicut-mimicking network based on the *expander decomposition* of Long and Saranurak [18] and the algorithm for a ϕ-expander with $\phi > 0$ outlined in Lemma 8. However, its return is not sufficiently small, and thus, we obtain a much smaller solution by applying it recursively.

MimickingExpander$(G; T, c)$. We let $n = |V(G)|$, $m = |E(G)|$, and $\phi^{-1} = 4rc^{Mr\log c}\log^3 n$, where the multicut-mimicking network returned by Lemma 8 has at most $kc^{Mr\log c}$ hyperedges for an expander with k terminals. When the submodule is called, we first decompose the vertex set $V(G)$ into the vertex partition $(V_1, \dots, V_s)$ so that the size of the cut of $(V_1, \dots, V_s)$ is at most $\phi m \log^3 n$ and each $\hat{G}[V_j]$ is a ϕ-expander for $j \in [s]$. There is a $p^{1+o(1)}$ time algorithm computing such a vertex partition [18], where $p = \sum_{e\in E} |e|$.

Every subinstance $(\hat{G}[V_j], T_j, c_j)$ is a ϕ-expander, where $c_j = \min\{c, |T_j|\}$ and T_j is the union of $T \cap V_j$ and anchored terminals in $\hat{G}[V_j]$. For each $(\hat{G}[V_j], T_j, c_j)$, we construct a multicut-mimicking network H_j by Lemma 8. Finally, we return the multicut-mimicking network H by gluing $H_1, \dots, H_s$. Precisely, we merge all restricted hyperedges having a common anchored terminal and remove all anchored terminals not in V. The following lemma summarizes the performance of this submodule, proved in the full version.

▶ **Lemma 9.** MimickingExpander$(G; T, c)$ *returns a multicut-mimicking network of* (G, T, c) *with* $(|T|c^{O(r\log c)} + (m/2))$ *hyperedges in* $(p^{1+o(1)} + |T|(c^{r\log c}\log n)^{O(rc)}m)$ *time.*

Overall algorithm. For an instance (G, T, c), we initialize $G_0 = G$ and obtain the multicut-mimicking network G_i by MimickingExpander$(G_{i-1}; T, c)$ for $i \in [\lceil \log m \rceil]$, inductively. Finally, we return $G_{\lceil \log m \rceil}$. This algorithm corresponds to Theorem 1. Details are in the full version.

▶ **Theorem 1.** *For* (G, T, c)*, we can compute a multicut-mimicking network of at most* $kc^{O(r\log c)}$ *hyperedges in* $p^{1+o(1)} + k(c^{r\log c}\log n)^{O(rc)}m$ *time, where* $k = |T|$ *and* $p = \sum_{e\in E} |e|$.

4 Bound for Minimal Instances

To complete the proof of Lemma 8 (and Theorem 1), we need to show that a minimal multicut-mimicking network for an expander (G, T, c) has at most $|T|c^{O(r\log c)}$ hyperedges. This section demonstrates it for not only expanders but also general instances. Precisely, we show the following theorem in this section. Here, r is the rank of G.

▶ **Theorem 10.** *Every minimal instance* (G, T, c) *has at most* $|T|c^{O(r\log c)}$ *hyperedges.*

In this section, we consider the scenario that every terminal in T has degree one in G. It is sufficient since the other case can be reduced to this scenario by inserting $c+1$ dummy terminals instead of each terminal t in T that are adjacent only to t. This reduction does not increase the rank or the parameter c while increasing the number of terminals by at most c times. However, this increase does not affect the asymptotic complexity in Theorem 10. The following explains the previous works and introduces the notions used in this section.

We broadly follow the approach of Wahlström [24]. He utilized the framework of Kratsch et al. [14] for multicut-mimicking networks in graphs without the parameter c. We incorporate the *unbreakable* concept to address the parameter c. Additionally, we slightly modify the *dense* concept used in the previous work to account for hypergraphs.

Unbreakable and dense. For a subinstance $(\hat{G}[X], T_X, c_X)$ with respect to $X \subset V(G)$, the terminal set T_X includes $T \cap X$ and up to two anchored terminals for each restricted hyperedge $e \in \partial_G X$. Hence, $|T_X| \leq 2|\partial_G X| + |T \cap X|$. We denote the value $2|\partial_G X| + |T \cap X|$ as $\mathsf{cap}_T(X; G)$, or $\mathsf{cap}_T(X)$ if the context is clear. Furthermore, we define the following:

- **Unbreakable:** An instance (G, T, c) is said to be d-unbreakable for $d > 0$ if $|T \cap X| \leq d$ for any vertex set $X \subseteq V(G)$ with $|T \cap X| \leq |T \setminus X|$ and $|\partial X| \leq c$.
- **Dense:** An instance (G, T, c) is said to be α-dense for $\alpha > 0$ if $|E(X)| \leq (\mathsf{cap}_T(X))^\alpha$ for any vertex set $X \subseteq V$ with $0 < |E(X)| \leq |E(V \setminus X)|$ and $|\partial X| \leq c$.

Wahlström [24] also used the concept of *dense* defining it based on the vertices instead of $E(\cdot)$. Since he addressed graphs, it was guaranteed that $|E(X)| = \Omega(|X|)$ by using connective assumption. However, this is not for hypergraphs. Thus, we slightly modify the definition.

In Section 4.2, we prove Theorem 10 for *unbreakable and dense* instances using the notions introduced in Section 4.1. Section 4.3 explains how to extend this proof for general instances.

4.1 Matroids and Representative Sets

We use the notion of matroids and representative sets as in the previous work, which is a generalization of the notion of linear independence in vector spaces. Formally, a matroid $(S, \mathcal{I})$ consists of a *universe set* S and an *independent set* $\mathcal{I} \subseteq 2^S$ with $\emptyset \in \mathcal{I}$ satisfying:

- If $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$, and
- If $A, B \in \mathcal{I}$ with $|A| < |B|$, then there exists $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

For a matroid $(S, \mathcal{I})$, its *rank* is the size of the largest set in $\mathcal{I}$. It is said to be *representable* if there is a matrix over a field whose columns are indexed by the elements of S such that: $F \subset S$ is in $\mathcal{I}$ if and only if the columns indexed by F are linearly independent over the field.

Representative sets. Kratsch et al. [14] introduced a framework for computing non-essential vertices using the notion of *representative sets*. We employ the framework. For this purpose, we introduce two operations: *truncation* and *direct sum* along with Lemma 11. For a matroid $(S, \mathcal{I})$ and an integer $r > 0$, the *r-truncation* of $(S, \mathcal{I})$ is defined as a matroid $(S, \mathcal{I}')$ such that $F \subseteq S$ is contained in $\mathcal{I}'$ if and only if $|F| \leq r$ and $F \in \mathcal{I}$. Note that an r-truncation has rank at most r. For matroids $\mathcal{M}_1, \ldots, \mathcal{M}_s$ over disjoint universes with $\mathcal{M}_i = (S_i, \mathcal{I}_i)$ for $i \in [s]$, their *direct sum* is defined as a matroid $(S, \mathcal{I})$ such that S is the union of all S_i, and a subset F of S is in $\mathcal{I}$ if and only if F can be decomposed into s disjoint subsets, each of which is independent in $\mathcal{M}_i$. The direct sum of matroids also satisfies the matroid axioms.

For a matroid $(S, \mathcal{I})$ and two subsets A, B of S, we say A *extends* B if $A \cap B = \emptyset$ and $A \cup B \in \mathcal{I}$. For $\mathcal{J} \subseteq 2^S$, a subset $\mathcal{J}^*$ of $\mathcal{J}$ is called a *representative set* if for any set $B \subseteq S$, there is a set $A^* \in \mathcal{J}^*$ that extends B when there is a set $A \in \mathcal{J}$ that extends B.

▶ **Lemma 11** ([14, Lemma 3.4]). *Let $\mathcal{M}_1 = (S_1, \mathcal{I}_1), \dots, \mathcal{M}_s = (S_s, \mathcal{I}_s)$ be matroids represented over the same finite field and with pairwise disjoint universe sets. Let $\mathcal{J}$ be a collection of sets containing one element from S_i for each $i \in [s]$. Then, some representative set of $\mathcal{J}$ in the direct sum of all matroids $\mathcal{M}_i$ has a size at most the product of the ranks of all $\mathcal{M}_i$.*

Uniform and (hyperedge) gammoids. We use two classes of representable matroids: *uniform matroids* and *gammoids*. They, along with their truncation and direct sum, are representable over any large field [7, 14, 17, 21, 22].

- **Uniform matroid:** For a set S and an integer $r > 0$, the *uniform matroid* on S of rank r is the matroid in which the universe set is S and the independent set consists of the sets containing at most r elements in S.
- **Gammoid:** For a directed graph $D = (V, A)$ and two subsets S and U of V, a *gammoid* defined on (D, S, U) is a matroid $(U, \mathcal{I})$ where $\mathcal{I}$ consists of the sets $X \subseteq U$ such that there are $|X|$ pairwise vertex-disjoint paths in D from S to X.

In this paper, we define and use *hyperedge gammoids* to handle hypergraphs.

- **Hyperedge gammoid:** For a hypergraph G and a terminal set $T \subset V(G)$, we consider a directed graph D^{split} defined as follows. First, we start from the undirected graph D^{split} of which the vertex set is $E(G)$ and $ee' \in E(D^{\mathsf{split}})$ for two e, e' in $E(G)$ (and $V(D^{\mathsf{split}})$) if and only if $e \cap e'$ is not empty. Then, we replace each undirected edge with two-way directed arcs. Furthermore, we insert a copy E_{sink} of $E(G)$ into $V(D^{\mathsf{split}})$. We call the copy in E_{sink} of a hyperedge $e \in E(G)$ a *sink-only copy* of e, and denote it by $\mathsf{sink}(e)$. We insert one-way arcs from e' to $\mathsf{sink}(e)$ on D^{split} for every $e' \in E(G)$ where $e' \cap e$ is not empty. The *hyperedge gammoid* of (G, T) is the gammoid on $(D^{\mathsf{split}}, \partial_G T \subset E(G), E(G) \cup E_{\mathsf{sink}})$.

Recall that a *path* of a hypergraph is defined as a sequence of hyperedges such that any two consecutive hyperedges contain a common vertex. If a path starts or ends at a hyperedge containing a terminal t, we say that it starts or ends at t to make the description easier. For a subset F of $E(G) \cup E_{\mathsf{sink}}$, let F_E be the set of hyperedges e of $E(G)$ where e or $\mathsf{sink}(e)$ is contained in F. Even if F contains both e and $\mathsf{sink}(e)$, e appears in F_E exactly once.

▶ **Observation 12.** *$F \subseteq E(G) \cup E_{\mathsf{sink}}$ is independent in the hyperedge gammoid of (G, T) if and only if there exist $|F|$ pairwise edge-disjoint paths from T to F_E in G with two exceptions:*

- *Two paths can end at $e \in E(G)$ if $e \in F$ and $\mathsf{sink}(e) \in F$, and*
- *One path can pass through e while another path ends at e if $e \notin F$ but $\mathsf{sink}(e) \in F$.*

4.2 Essential Hyperedges in Unbreakable and Dense Instances

Assume that (G, T, c) is d-unbreakable and α-dense with $c \leq d \leq |T|$ and $\alpha \geq 35r \log d$, where r is the rank of G. In this section, we show that (G, T, c) contains at most $|T|d^{\alpha-1}$ essential hyperedges which directly implies Theorem 10 for $O(c)$-unbreakable and $\Theta(r \log c)$-dense instances. Precisely, if there are more than $|T|d^{\alpha-1}$ hyperedges, then at least one is non-essential, and thus, the instance is not minimal. Here, $k = |T|$ and r is the rank of G.

The following matroids would be a witness for our claim with $i_0 = 30r$:

- One uniform matroid on E of rank $(\kappa + c)$, where $\kappa = (d/2)^{\alpha - i_0 - 2}$,
- One $(k + c + 1)$-truncation $\mathcal{M}_0$ of the hyperedge gammoid of (G, T), and
- i_0 copies $\mathcal{M}_1, \dots, \mathcal{M}_{i_0}$ of the $(d + c + 1)$-truncation of the hyperedge gammoid of (G, T),

For a hyperedge e, its appearance in the universe sets of the uniform matroid on E is denoted by e^{u}. In the universe set of $\mathcal{M}_i$ for $i \in [0, i_0]$, e_i and $\mathsf{sink}_i(e)$ denote the hyperedge and its sink-only copy, respectively. Note that sink-only copies have no outgoing arc in D^{split}, where the hyperedge gammoid is defined on the directed graph D^{split}, refer to Section 4.1.

Let $\mathcal{M}$ denote the direct sum of the (i_0+2) matroids described above. For a hyperedge $e \in E(G)$, let $J(e) = \{e^{\mathsf{u}}\} \cup \{\mathsf{sink}_0(e), \ldots, \mathsf{sink}_{i_0}(e)\}$. Then, we let $\mathcal{J}^*$ be the representative set of $\{J(e) \mid e \in E(G)\}$ outlined in Lemma 11. The set $\mathcal{J}^*$ consists of all essential hyperedges.

▶ **Lemma 13.** *$\mathcal{J}^*$ contains all $J(\bar{e})$ where $\bar{e}$ is an essential hyperedge for (G, T, c).*

Proof. We fix an essential hyperedge $\bar{e}$ and show that $J(\bar{e})$ is in $\mathcal{J}^*$. To do this, we construct an independent set in $\mathcal{M}$ extended by $J(\bar{e})$, but not extended by $J(e)$ for any hyperedge $e \neq \bar{e}$. Let $\mathcal{T}$ be a terminal partition with $\mathsf{min\text{-}cut}_G(\mathcal{T}) \leq c$ such that every minimum multiway cut of $\mathcal{T}$ includes $\bar{e}$. We fix a minimum multiway cut F of $\mathcal{T}$ in G, and let $(V_0, \ldots, V_s)$ be the vertex partition according to $G \setminus F$. We assume that the component V_0 maximizes $|T \cap V_0|$ among $V_0, \ldots, V_s$, and V_1 maximizes $|E(V_1)|$ among $V_1, \ldots, V_s$. Additionally, the other components $V_2, \ldots, V_s$ are sorted in the decreasing order of $\mathsf{cap}_T(\cdot)$ values. Let E_{small} denote the union of $E(V_{i_0+1}), E(V_{i_0+2}), \ldots, E(V_s)$.

Then we consider the following sets whose union will be a witness showing that $J(\bar{e})$ is in any representative set of $\{J(e) \mid e \in E(G)\}$. Let A^{u} denote the subset $\{e^{\mathsf{u}} \mid e \in (E_{\mathsf{small}} \cup F) \setminus \{\bar{e}\}\}$ of the universe set of the uniform matroid on E of rank $\kappa + c$. Let A_i^{g} be the subset $\{e_i \mid e \in F \cup \partial(T \cap V_i)\}$ of the universe set of $\mathcal{M}_i$ for $i \in [0, i_0]$. We need to demonstrate three assertions: **(i)** A^{u} is extended by $\{(\bar{e})^{\mathsf{u}}\}$ in the uniform matroid on E of rank $\kappa + c$, **(ii)** A_i^{g} is extended by $\{\mathsf{sink}_i(\bar{e})\}$ in $\mathcal{M}_i$ for $i \in [0, i_0]$, and **(iii)** any $J(e)$ with $e \neq \bar{e}$ cannot extend $A = A^{\mathsf{u}} \cup (\cup_{i \in [0, i_0]} A_i^{\mathsf{g}})$ in $\mathcal{M}$. By combining these assertions, we can conclude that A is extended in $\mathcal{M}$ by $J(\bar{e})$ only, which completes the proof of Lemma 13.

Assertion (i). Recall that the size of F is at most c. Then A^{u} is extended by $\{(\bar{e})^{\mathsf{u}}\}$ in the uniform matroid on E of rank $\kappa + c$ if $|E_{\mathsf{small}}| \leq \kappa$ because $\bar{e}$ is not in $(E_{\mathsf{small}} \cup F) \setminus \{\bar{e}\}$.

We first show that $\sum_{j \in [s]} \mathsf{cap}_T(V_j) \leq (3d + 2rc)$. Note that every component V_j has at most d terminals of T for $j \in [s]$ because (G, T, c) is d-unbreakable. Precisely, if a component V_j has more than d terminals, then V_0 also contains more than d terminals since we chose V_0 as containing the most terminals, and thus, the d-unbreakable property is violated from $|V_j \cap T|, |T \setminus V_j| > d$. Let x be the smallest index in $[s]$ such that $\sum_{j \in [x]} |T \cap V_j| > d$. By choosing x in this manner, the total size of $T \cap V_j$ for all indices $1 \leq j < x$ is at most d, and for all indices $j > x$ is also at most d due to the d-unbreakable property. Due to $|T \cap V_x| \leq d$, we have $\sum_{j \in [s]} |T \cap V_j| \leq 3d$. Moreover, since each hyperedge in F can intersect at most r components and $\mathsf{cap}_T(V_j) = 2|\partial_G V_j| + |T \cap V_j|$, we have $\sum_{j \in [s]} \mathsf{cap}_T(V_j) \leq (3d + 2rc)$.

Now we focus on the components $V_{i_0+1}, \ldots, V_s$ as follows. Since (G, T, c) is α-dense, $|E(V_j)| \leq (\mathsf{cap}_T(V_j))^\alpha$ for $j \in [2, s]$ by the α-dense property.[1] Thus, the ordering $\mathsf{cap}_T(V_2) \geq \cdots \geq \mathsf{cap}_T(V_s)$ implies that $\mathsf{cap}_T(V_j) \leq (3d + 2rc)/(j-1)$. Therefore, we have

$$\begin{aligned} |E_{\mathsf{small}}| = \sum_{j \in [i_0+1, s]} |E(V_j)| &\leq \sum_{j \in [i_0+1, s]} \left(\frac{3d + 2rc}{i_0}\right)^\alpha \leq \sum_{j \in [i_0+1, s]} \left(\frac{d}{4}\right)^\alpha \\ &\leq (d/4)^\alpha \cdot rc \leq (d/2)^{\alpha+1} \cdot (1/2)^{\alpha-1} r \leq (d/2)^{\alpha - i_0 - 2} = \kappa. \end{aligned}$$

[1] $|E(V_j)| = \min\{|E(V_j)|, |E(V \setminus V_j)|\} \leq \mathsf{cap}_T(V_j)^\alpha$ for $j \in [2, s]$.

The inequalities follow from $i_0 = 30r$, $s \leq rc$, and the assumption we made at the beginning of this subsection: $c \leq d \leq k$ and $\alpha \geq 35r \log d \geq (i_0 + 2) \log d$.[2] It implies $|A^{\mathsf{u}}| \leq \kappa + c - 1$ due to $(\bar{e})^{\mathsf{u}} \notin A^{\mathsf{u}}$. Therefore, $\{(\bar{e})^{\mathsf{u}}\}$ extends A^{u} in the uniform matroid on E of rank $\kappa + c$.

Sketch of Assertions (ii-iii). Details are in the full version. Briefly, **Assertions (ii-iii)** holds since there are $|F| + 1$ pairwise edge-disjoint paths from $T \setminus V_i$ to F in G for each component V_i if we allow using $\bar{e} \in F$ twice while there is no path from $T \setminus V_i$ to $E(V_i)$ excluding F. Recall that $\bar{e}$ is essential. Therefore, $\{\mathsf{sink}_i(e)\}$ extends A_i^{g} if and only if $e = \bar{e}$ by Observation 12 along with the Menger's theorem for hypergraphs [26]. ◀

Lemma 14 holds by Lemma 11 and Lemma 13. The detailed proof is in the full version.

▶ **Lemma 14.** *If (G, T, c) is d-unbreakable and α-dense with $\alpha \geq 35r \log d$, $c \leq d \leq k$, and $m > kd^{\alpha-1}$, then there is a non-essential hyperedge.*

4.3 Non-Essential Hyperedge in a General Instance

In this section, we show that a minimal multicut-mimicking network of (G, T, c) has at most $|T|c^{O(r \log c)}$ hyperedges. For this achievement, we start by assuming that (G, T, c) has more than $|T|c^{\Omega(r \log c)}$ hyperedges, and find a subinstance that has a non-essential hyperedge. Note that the obtained hyperedge is also non-essential in (G, T, c) by Lemma 3, and thus, (G, T, c) is not minimal by Lemma 2. Note that we can find a $5c$-unbreakable subinstance (G', T', c) of $|T'|c^{\Omega(r \log c)}$ hyperedges [10]. Details are in the full version. In the following, we suppose that (G, T, c) is a $5c$-unbreakable with $|T|c^{\Omega(r \log c)}$ hyperedges.

We recursively find a subinstance until it satisfies the conditions in Lemma 14. Then it has a non-essential hyperedge, furthermore, it is also non-essential in the original instance. Note that constructing a subinstance might increase the size of a hyperedge by inserting two anchored terminals for a restricted hyperedge. However, it increases the hyperedge size only if the hyperedge has less than two terminals. Additionally, once increased hyperedge has two (anchored) terminals. Thus, the rank is increased by at most one even if we obtain a subinstance recursively. We fix r as the rank of the original instance to avoid confusion.

Construction of non-minimal instance. We suppose that (G, T, c) is d-unbreakable with $d = \min\{5c, |T|\}$. Then we show that if G has more than $|T|d^{\alpha(c)-1}$ hyperedges, (G, T, c) has a non-essential hyperedge by inductively along m, where $\alpha(c) = 35(r + 2) \log(5c)$. Recall that r is a fixed constant so that the rank of (G, T, c) is at most $r + 1$, and α is the constant derived from c only. For simplicity, we use α to denote $\alpha(c)$ when the context is clear. If (G, T, c) is α-dense, then it has a non-essential hyperedge by Lemma 14.

When (G, T, c) is not α-dense, there is a witness vertex set $X \subseteq V$ with $0 < |E(X)| \leq |E(V \setminus X)|$, $|\partial X| \leq c$, and $|E(X)| > (\mathsf{cap}_T(X))^{\alpha}$.[3] If $|X \cap T| > |T \setminus X|$, we *replace* X with $V \setminus X$. Note that the following inequalities still hold: $|\partial X| \leq c$, $|E(X)| > (\mathsf{cap}_T(X))^{\alpha}$, and $|E(V \setminus X)| > 0$. The first one holds since $\partial X = \partial(V \setminus X)$, and the second one holds since $|T \cap X|$ and $\mathsf{cap}_T(X)$ are decreased by the replacement while $|E(X)|$ is increased. The last holds since we chose X so that $E(X), E(V \setminus X) \neq \emptyset$. Additionally, the size of $T \cap X$ is at most d since our instance is d-unbreakable. We move to the subinstance $(\hat{G}[X], T_X, c_X)$ with respect to X, where $c_X = \min\{c, |T_X|\}$ and T_X is the union of terminals $T \cap X$ and the anchored terminals. Recall that the size of T_X is at most $\mathsf{cap}_T(X; G) = |T \cap X| + 2|\partial X| \leq d + 2c$. Lemma 15, proved in the full version, ensures the safeness of this inductive proof.

[2] Precisely, these assumption give us $r \cdot \left(\frac{1}{2}\right)^{\alpha-1} \leq \left(\frac{d}{2}\right)^{-i_0-3}$ which implies the last inequality.

[3] (G, T, c) is α-dense if $|E(Y)| \leq (\mathsf{cap}_T(Y))^{\alpha}$ for any $Y \subseteq V$ with $0 < |E(Y)| \leq |E(V \setminus Y)|$ and $|\partial Y| \leq c$.

▶ **Lemma 15.** *$(\hat{G}[X], T_X, c_X)$ is d_X-unbreakable with $d_X = \min\{5c_X, |T_X|\}$. Additionally, $\hat{G}[X]$ has more than $|T_X|d_X^{\alpha'-1}$ hyperedges but less than $|E(G)|$, where $\alpha' = \alpha(c_X)$.*

We recursively obtain a subinstance until it becomes α-dense. Note that k, d, m, and α change during the recursion while the rank is always at most $r+1$, where k and m denote the number of terminals and hyperedges, respectively, in the current instance. However, Lemma 15 guarantees that $m > kd^{\alpha-1}$ holds at each step. Moreover, m is strictly decreased. Thus, we always reach a d-unbreakable and α-dense instance satisfying the conditions of Lemma 14, and it has a non-essential hyperedge. It is easy to show that the hyperedge is also non-essential in the original instance by applying Lemma 3 recursively.

Therefore, a d-unbreakable instance (G, T, c) with $m > |T|d^{\alpha(c)-1}$ and $d = \min\{5c, |T|\}$ has a non-essential hyperedge. This section proves Theorem 10.

▶ **Theorem 10.** *Every minimal instance (G, T, c) has at most $|T|c^{O(r \log c)}$ hyperedges.*

References

1 Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: A survey. *Integration*, 19(1-2):1–81, 1995. doi:10.1016/0167-9260(95)00008-4.

2 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 910–928, 2019. doi:10.1109/FOCS.2019.00059.

3 Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P. Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 1206–1225, 2021. doi:10.1137/1.9781611976465.74.

4 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. *ACM Transactions on Algorithms (TALG)*, 14(3):1–15, 2018. doi:10.1145/3199607.

5 Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018. doi:10.1137/18M1163865.

6 Yu Chen and Zihan Tan. On $(1+\varepsilon)$-approximate flow sparsifiers. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 1568–1605, 2024. doi:10.1137/1.9781611977912.63.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4(8). Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.

9 Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC 2016)*, pages 9–17, 2016. doi:10.1145/2933057.2933114.

10 Han Jiang, Shang-En Huang, Thatchaphol Saranurak, and Tian Zhang. Vertex sparsifiers for hyperedge connectivity. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA 2022)*, pages 70:1–70:13, 2022. doi:10.4230/LIPICS.ESA.2022.70.

11 Wenyu Jin and Xiaorui Sun. Fully dynamic *s*-*t* edge connectivity in subpolynomial time. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 861–872. IEEE, 2022.

12 Jörg Hendrik Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Higher-order segmentation via multicuts. *Computer Vision and Image Understanding*, 143:104–119, 2016. doi:10.1016/J.CVIU.2015.11.005.

13 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Spectral hypergraph sparsifiers of nearly linear size. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 1159–1170, 2022.

14 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM*, 67(3):1–50, 2020. doi:10.1145/3390887.

15 Robert Krauthgamer and Ron Mosenzon. Exact flow sparsification requires unbounded size. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2354–2367, 2023. doi:10.1137/1.9781611977554.CH91.

16 Yang P Liu. Vertex sparsification for edge connectivity in polynomial time. In *Proceedings of the 14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, pages 83:1–83:15, 2023. doi:10.4230/LIPICS.ITCS.2023.83.

17 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Transactions on Algorithms (TALG)*, 14(2):1–20, 2018. doi:10.1145/3170444.

18 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *Proceedings of the 63rd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 1002–1010, 2022. doi:10.1109/FOCS54457.2022.00098.

19 Anand Louis and Yury Makarychev. Approximation algorithms for hypergraph small set expansion and small set vertex expansion. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 339, 2014.

20 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. doi:10.1016/J.TCS.2005.10.007.

21 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009. doi:10.1016/J.TCS.2009.07.027.

22 James Oxley. Matroid theory. In *Handbook of the Tutte Polynomial and Related Topics*, pages 44–85. Chapman and Hall/CRC, 2022.

23 Muhammet Mustafa Ozdal and Cevdet Aykanat. Hypergraph models and algorithms for data-pattern-based clustering. *Data Mining and Knowledge Discovery*, 9:29–57, 2004. doi:10.1023/B:DAMI.0000026903.59233.2A.

24 Magnus Wahlström. Quasipolynomial multicut-mimicking networks and kernels for multiway cut problems. *ACM Transactions on Algorithms (TALG)*, 18(2):1–19, 2022. doi:10.1145/3501304.

25 Zi-Ke Zhang and Chuang Liu. A hypergraph model of social tagging networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(10):P10005, 2010.

26 Alexander Aleksandrovich Zykov. *Hypergraphs*, volume 29(6). IOP Publishing, 1974.

On HTLC-Based Protocols for Multi-Party Cross-Chain Swaps

Emily Clark
University of California, Riverside, CA, USA

Chloe Georgiou
University of California, Riverside, CA, USA

Katelyn Poon
University of California, Riverside, CA, USA

Marek Chrobak
University of California, Riverside, CA, USA

Abstract

In his 2018 paper, Herlihy introduced an atomic protocol for multi-party asset swaps across different blockchains. Practical implementation of this protocol is hampered by its intricacy and computational complexity, as it relies on elaborate smart contracts for asset transfers, and specifying the protocol's steps on a given digraph requires solving an $\mathbb{NP}$-hard problem of computing longest paths. Herlihy left open the question whether there is a simple and efficient protocol for cross-chain asset swaps in arbitrary digraphs. Addressing this, we study *HTLC-based protocols*, in which all asset transfers are implemented with standard hashed time-lock smart contracts (HTLCs). Our main contribution is a full characterization of swap digraphs that have such protocols, in terms of so-called reuniclus graphs. We give an atomic HTLC-based protocol for reuniclus graphs. Our protocol is simple and efficient. We then prove that non-reuniclus graphs do not have atomic HTLC-based swap protocols.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed computing, blockchain, asset swaps

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.22

Related Version All proofs omitted in this extended abstract can be found in the full version.
Full Version: `https://doi.org/10.48550/arXiv.2403.03906` [5]

Funding Research partially supported by National Science Foundation grant CCF-2153723.

1 Introduction

In 2018, Herlihy [9] introduced a model for multi-party asset swaps across different blockchains, where an asset swap is represented by a strongly connected directed graph, with each vertex corresponding to one party and each arc representing a pre-arranged asset transfer between two parties. The goal is to design a protocol to implement the transfer of all assets. The protocol must guarantee, irrespective of the behavior of other parties, that each honest party will end up with an outcome that it considers acceptable. The protocol should also discourage cheating, so that any coalition of parties cannot improve its outcome by deviating from the protocol. These two conditions are called *safety* and *strong Nash equilibrium*, respectively. A protocol that satisfies these conditions is called *atomic*.

In this model, Herlihy [9] developed an atomic protocol for asset swaps in arbitrary strongly connected digraphs. While this result is a significant theoretical advance, its practical implementation is hampered by its intricacy and high computational complexity, as it relies on elaborate smart contracts for asset transfers, and specifying the protocol's steps on a given digraph requires solving an $\mathbb{NP}$-hard problem of computing longest paths. It also uses a cryptographic scheme with nested digital signatures that may reveal the graph's topology to all parties, raising concerns about privacy.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 22; pp. 22:1–22:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Herlihy [9] also mentions a simpler protocol that uses only standard smart contracts called *hashed time-lock contracts* (HTLC's), that require only one secret/hashlock pair and a time-out mechanism (see [19, 16]). This protocol, however, works correctly only for special types of digraphs that we call *bottleneck graphs.* This raises a natural question, already posed in [9]: *Is there a simple and efficient protocol for multi-party asset swaps that is atomic and works on all strongly connected digraphs?*

Our contributions. Motivated by this question, we study *HTLC-based protocols*, which are allowed to exchange assets only via HTLC's. As it turns out, the class of digraphs that have such protocols is much broader than bottleneck digraphs; in fact, we give a complete characterization of digraphs that admit HTLC-based protocols. We call them *reuniclus graphs.* Roughly, a reuniclus graph can be thought of as a tree of biconnected components, each being an induced bottleneck subgraph. In this terminology, our main contribution can be stated as follows:

▶ **Theorem 1.** *A swap digraph G has an atomic HTLC-based protocol if and only if G is a reuniclus digraph.*

The sufficiency condition is proved by providing an atomic HTLC-based protocol for reuniclus digraphs. The protocol itself is simple and efficient. Also, testing whether a given graph is a reuniclus graph and, if it is, computing the specification of the protocol for each party can be accomplished in linear time. (The key ingredient is the algorithm from [12] that can be used to recognize bottleneck graphs.) Our most technically challenging contribution is the proof of the necessity condition. This requires showing that the atomicity assumption implies some structural properties of the underlying graph. By carefully exploiting this approach, we prove that each digraph with an atomic protocol must have the reuniclus structure.

Our asset-swap model is in fact a slight generalization of the one in [9], as it uses a relaxed definition of the preference relation, which allows each party to customize some of their preferences.

Related work. The problem of securely exchanging digital products between untrustful parties has been studied since 1990s under the name of *fair exchange.* As simultaneous exchange is not feasible in a typical electronic setting, protocols for fair exchange rely on a trusted party – see for example [15, 7, 3, 1, 2]. In the model from [9], smart contracts play the role of trusted parties.

With users now holding assets on a quickly growing number of different blockchains, cross-chain interoperability tools became necessary to allow these users to trade their assets. An atomic swap concept was one of the proposed tools to address this issue. The concept itself and some early implementations of asset-swap protocols (see, for example [18]) predate the work of Herlihy [9].

In recent years there has been intensive research activity on asset-swap protocols. The preference relation of the participants in the model from [9] is very rudimentary, and some refinements of this preference model were studied in [4, 11]. Some proposals [14, 20] address the issue of “ grieving”, when one party needs to wait for the counter-party to act, while its assets are locked and unaccessible. Other directions of study include investigations of protocol's time and space complexity [11], privacy issues [6], security enhancements [13], and generalizations of swaps to more complex transactions [10, 17, 8].

2 Multi-Party Asset Swaps

The *multi-party asset swap problem* we address is this: There is a set V of parties, each with a set of assets that it wishes to exchange for some other assets. Suppose that there is a way to reassign assets from each current owner to their new owner in such a way that each party would receive exactly their desired assets. This reassignment is called a *multi-party asset swap*. The goal is now to arrange the transfers of these assets. The challenge is that some parties may deviate from the protocol, attempting to improve their outcome, or even behave irrationally. To address this, the asset-swap protocol must satisfy the following safety property in addition to correctness: an outcome of any honest party (that follows the protocol) must be guaranteed to be acceptable (not worse than its initial holdings), even if other parties deviate from the protocol or collaboratively attempt to cheat.

Let $G = (V, A)$ be a digraph with vertex set V and arc set A, without self-loops or parallel arcs. By N_v^{in} we denote the set of in-neighbors of v, by A_v^{in} its set of incoming arcs, N_v^{out} are the out-neighbors of v and A_v^{out} are its outgoing arcs. Other graph notation and terminology is standard.

Clearing service. We assume the existence of a *market clearing service*, where each party submits its proposed exchange (the collections of its current and desired assets). If a swap is possible, the clearing service constructs a digraph $G = (V, A)$ representing this swap. Each arc (u, v) of G represents the intended transfer of one asset from its current owner u (the seller) to its new owner v (the buyer). For simplicity, we assume that any party can transfer only one asset to any other party, and we identify assets with arcs, so (u, v) denotes both an arc of G and the asset of u to be transferred to v. The service ensures that G satisfies the assumptions of the swap protocol, and it informs each party of the protocol's steps. Importantly, we *do not* assume that the parties trust the market clearing service.

Secrets and hashlocks. We allow each party v to create a *secret value* s_v, and convert it into a *hashlock value* $h_v = H(s_v)$, where $H()$ is a one-way permutation. The value of s_v is secret, meaning that no other party has the capability to compute s_v from h_v. The hashlock values can be made public.

Hashed time-lock contracts (HTLCs). Asset transfers are realized with smart contracts, which are simply pieces of code running on a blockchain. The contracts used in our model are called *hashed time-lock contracts*, or *HTLCs*, for short, and are defined as follows: Each contract is associated with an arc (u, v) of G, and is used to transfer the asset (u, v) from u to v. It is created by u, with u providing it with the asset, timeout value τ, and a hashlock value h. Once this contract is created, the possession of the asset is transferred from u to the contract. The counter-party v can access the contract to verify its correctness; in particular, it can learn the hashlock value h. There are two ways in which the asset can be released: (1) One, v can claim it. To claim it successfully, v must provide a value s such that $H(s) = h$ not later than at time τ. When this happens, the smart contract transfers the asset to v, and it gives s to u. (2) Two, the contract can expire. As soon as the current time exceeds τ, and if the asset has not been claimed, the contract returns the asset to u.

Further overloading notation and terminology, we will also refer to the contract on arc (u, v) as "contract (u, v)". If this contract has hashlock h_x of a party x (where x may be different from u and v), we will say that it is *protected by hashlock h_x* or simply *protected by party x*.

Protocols. In an execution of $\mathbb{P}$ there is no guarantee that all parties actually follow $\mathbb{P}$. When we refer to an *honest* or *conforming* party u, we mean that u follows $\mathbb{P}$, except when it can infer that not all parties follow $\mathbb{P}$. From that point on, u may behave arbitrarily (but still rationally).

In an *HTLC-based protocol*, all asset transfers are implemented with HTLCs, and no other interaction between the parties is allowed. Each party can create one secret/hashlock pair. These hashlock values are distributed via smart contracts (or can simply be made public).

Outcomes. For each party v, *v's outcome* associated with an execution of a protocol $\mathbb{P}$ is specified by the sets of assets that are relinquished and acquired by v. Thus such an outcome is a pair $\omega = \left\langle \omega^{in} \,|\, \omega^{out} \right\rangle$, where $\omega^{in} \subseteq A^{in}_v$ and $\omega^{out} \subseteq A^{out}_v$. To reduce clutter, instead of arcs, in $\left\langle \omega^{in} \,|\, \omega^{out} \right\rangle$ we can list only the corresponding in-neighbors and out-neighbors of v; for example, instead of $\langle \{(x, v), (y, v)\} \,|\, \{(v, z)\} \rangle$ we will write $\langle x, y \,|\, z \rangle$.

An outcome $\omega = \left\langle \omega^{in} \,|\, \omega^{out} \right\rangle$ of some party u is called *acceptable* if in this outcome u retains all its own assets or it gains all incoming assets. That is, either $\omega^{in} = A^{in}_v$ or $\omega^{out} = \emptyset$. The following two types of outcomes are particularly significant: $\textsc{Deal}_v = \left\langle A^{in}_v \,|\, A^{out}_v \right\rangle$ represents an outcome where all prearranged asset transfers involving v are completed, and $\textsc{NoDeal}_v = \langle \emptyset \,|\, \emptyset \rangle$ represents an outcome where none of the prearranged asset transfers involving v is completed. We will skip the subscript v in these notations whenever v is understood from context[1].

For a set C of parties, its set A^{in}_C of incoming arcs consists of arcs (u, v) with $u \notin C$ and $v \in C$. The set A^{out}_C of outgoing arcs is defined analogously. With this, the concept of outcomes and its properties extend naturally to sets of parties ("coalitions"). For example, an outcome of C is *acceptable* if it either contains all incoming arcs of C or does not contain any outgoing arcs of C.

The preference relation. A *preference relation* of a party v is a partial order on the set of all outcomes for v that satisfies the following three properties: **(p1)** If $\omega^{in}_1 \subseteq \omega^{in}_2$ and $\omega^{out}_1 \supseteq \omega^{out}_2$, then ω_2 is preferred to ω_1 ; **(p2)** If ω is unacceptable then $\textsc{NoDeal}$ is preferable to ω; **(p3)** $\textsc{Deal}$ is better than $\textsc{NoDeal}$. These are natural: (p1) says that it is better to receive more assets and relinquish fewer assets, and without (p3) v would have no incentive to participate in the protocol. The preference relation captures rational behavior, leading to the definition of Nash equilibrium property, below.

Protocol properties. Following [9], we define the following properties of a swap protocol $\mathbb{P}$:

Liveness: $\mathbb{P}$ is *live* if each party ends up in $\textsc{Deal}$, providing that all parties follow $\mathbb{P}$.

Safety: $\mathbb{P}$ is *safe* if each honest party ends up in an acceptable outcome, independently of the behavior of other parties.

Strong Nash Equilibrium: $\mathbb{P}$ has the *strong Nash equilibrium property* if for any set C of parties, if all parties outside C follow $\mathbb{P}$ then the parties in C cannot improve the outcome of C by deviating from $\mathbb{P}$.

Atomicity: $\mathbb{P}$ is called *atomic* if it's live, safe, and has the strong Nash equilibrium property.

The lemma below is a mild extension of the one in [9]. The point of the lemma is that, in Herlihy's preference model, the strong Nash equilibrium property comes for free. The strong connectivity assumption is necessary for the safety property to hold, see [9]. (See the full paper for the easy proof.)

[1] Herlihy [9] defines other types of outcomes: $\textsc{Discount}$, $\textsc{FreeRide}$ and $\textsc{Underwater}$, but these are not essential – see the full version of the paper.

▶ **Lemma 2.** *Assume that digraph G is strongly connected. If a protocol $\mathbb{P}$ is live and safe then $\mathbb{P}$ is atomic.*

3 An Atomic Protocol for Reuniclus Digraphs

In this section we give an atomic asset-swap protocol for reuniclus digraphs. First, in Section 3.1, we describe an atomic protocol for bottleneck digraphs called Protocol BDP. This protocol is mostly equivalent to the simplified protocol from [9, Section 4.6]. We include it here, because it serves as a stepping stone to our full protocol for reuniclus digraphs that is presented in Section 3.2.

3.1 Protocol BDP for Bottleneck Digraphs

Protocol BDP for leader ℓ:

- *At time* 0*:* Create a secret s_ℓ and compute $h_\ell = H(s_\ell)$. For each arc (ℓ, v), create contract with hashlock h_ℓ and timeout $\tau_{\ell v} = D^* + D_v^+$.
- *At time* D^**:* Claim all incoming assets using secret s_ℓ.

Protocol BDP for a follower u:

- *At time* D_u^-*:* For each arc (u, v), create contract with hashlock h and timeout $\tau_{uv} = D^* + D_v^+$.
- *At time* $D^* + D_u^+$*:* Let s be the secret obtained from the contract for some claimed outgoing assets. Use s to claim all incoming assets.

Figure 1 Protocol BDP, for the leader on the left, and for the followers on the right. Each bullet-point step takes one time unit. In the description we tacitly assume that u aborts if it detects any deviation from the protocol.

A vertex v in a digraph G is called a *bottleneck vertex* if it belongs to each cycle of G. If G is strongly connected and has a bottleneck vertex then we refer to G as a *bottleneck digraph.*

We now describe Protocol BDP for a bottleneck digraph G. One bottleneck vertex of G is designated as the *leader.* This leader, denoted ℓ, creates its secret/hashlock pair (s_ℓ, h_ℓ). The other vertices are called *followers.* Protocol BDP has two phases. The first phase, initiated by ℓ, creates all contracts. Each follower waits for all the incoming contracts to be created, and then creates the outgoing contracts. For followers, the timeout values for all incoming contracts are strictly larger than the timeout values for all outgoing contracts. In the second phase the assets are claimed, starting with ℓ claiming its incoming assets. Now the process proceeds backwards. For each follower v, when any of its outgoing assets is claimed, v learns the secret value s_ℓ, and it can now claim its incoming assets.

The detailed description of this protocol is given in Figure 1. In the protocol, D_v^- denotes the *maximum distance from* ℓ *to* v, defined as the maximum length of a simple path from ℓ to v. In particular, $D_\ell^- = 0$. The values D_y^- are used in contract creation times. By D^* we denote the maximum length of a simple cycle in G, so $D^* = \max_{z \in N_\ell^{in}} D_z^- + 1$.

In the timeout values, the notation D_v^+ is the maximum distance from v to ℓ. Naturally, we have $\max_{z \in N_\ell^{out}} D_z^+ + 1 = D^*$. Note that, for each v, the timeouts of all incoming contracts (u, v) are equal to $D^* + D_v^+$, exactly when v claims them. Also, if $v \neq \ell$ then $D^* + D_v^+$ is larger than the timeout $D^* + D_w^+$ of each outgoing contract (v, w).

▶ **Theorem 3.** *If G is a bottleneck digraph, then Protocol BDP is an atomic swap protocol for G.*

The safety property should be intuitive: The leader protects its outgoing assets, so it will lose these assets only if it first claims its incoming assets. If a follower loses an outgoing asset, it has at least one time unit to claim its incoming assets. The formal proof is similar (in fact, easier) to that for reuniclus graphs and omitted. (See the full version.)

3.2 Protocol RDP for Reuniclus Digraphs

Reuniclus digraphs. A strongly connected digraph G is called a *reuniclus digraph* if there are vertices $b_1, b_2, ..., b_p \in G$, induced subgraphs $G_1, G_2, ..., G_p$ of G, and a rooted tree $\mathcal{K}$ whose nodes are $b_1, b_2, ..., b_p$, such that: (rg1) Each digraph G_j is a bottleneck subgraph, with b_j being its bottleneck vertex. We call G_j a *bottleneck component* of G and the *home component* of b_j. (rg2) If $i \neq j$, then $G_i \cap G_j = \{b_j\}$ if b_i is the parent of b_j in $\mathcal{K}$, and $G_i \cap G_j = \emptyset$ otherwise.

We refer to $\mathcal{K}$ as the *control tree* of G. (See Figure 2 for an example.) We extend the tree terminology to relations between bottleneck components, or between bottleneck vertices and components, in a natural fashion. Intuitively, a reuniclus graph G can be divided into bottleneck components. Overlaps are allowed only between two components if one is the parent of the other in the control tree $\mathcal{K}$, in which case the overlap is just a single vertex that is the bottleneck of the child component.

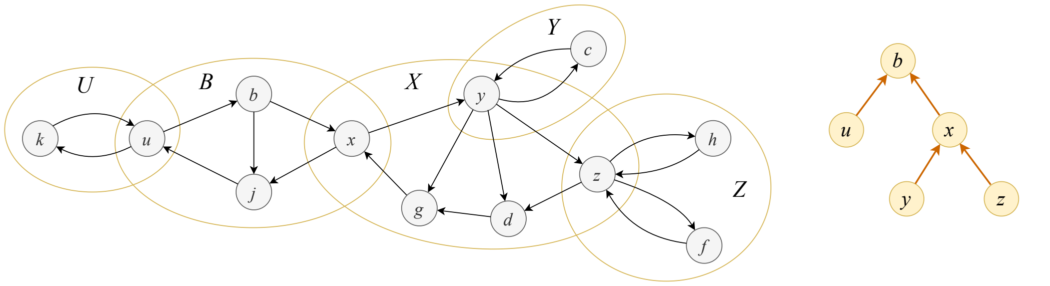


Figure 2 An example of a reuniclus graph (left) and its control tree (right). The bottleneck components (circled) are B, U, X, Y and Z. Their designated bottleneck vertices are b, u, x, y and z.

From the definition, the set of all bottleneck vertices in G forms a feedback vertex set of G. These bottleneck vertices are articulation vertices of G. Each bottleneck component may consist of several biconnected components that share the same bottleneck vertex.

Protocol RDP. Protocol RDP can be thought of as a hierarchical extension of Protocol BDP. Each bottleneck vertex b_j is called the leader of G_j. It creates a secret/hashlock pair (s_j, h_j), and its hashlock h_j is used to transfer assets within G_j, while the transfer of assets in the descendant components of b_j is "delegated" to the children of b_j in $\mathcal{K}$. We assume that the root component of $\mathcal{K}$ is G_1, and its bottleneck b_1 is called the *main leader*, denoted also by ℓ. All non-leader vertices are called *followers*.

Protocol RDP has two phases: contract creation and asset claiming. In the first phase, at time 0 all leaders create the outgoing contracts within their home bottleneck components. Then the contracts are propagated within the bottleneck components, to some degree independently; except that each leader b_j creates its outgoing contracts in its parent component G_i only after all its incoming contracts, *both in G_i and G_j*, are created. This ensures that at that time all contracts in its descendant components will be already created.

Protocol RDP for a leader $b_j \in G_i \cap G_j$:

- *At time* 0*:* Generate secret s_j and compute $h_j = H(s_j)$. For each arc (b_j, v) in G_j, create contract with hashlock h_j and timeout $\tau_{b_j v} = B^* + D^+_v$.
- *At time* $B^-_{b_j}$*:* For each arc (b_j, v) in G_i create its contract with hashlock h and timeout $\tau_{b_j v} = B^* + D^+_v$.
- *At time* $B^* + D^+_{b_j}$*:* Claim all incoming assets, using secret s in G_i and secret s_j in G_j.

Figure 3 Protocol RDP for a sub-leader b_j, namely the bottleneck vertex of G_j that also belongs to its parent component G_i. Recall that B^-_u denotes the maximum distance from some leader to u along a path that satisfies conditions (i)-(iii), and that $B^* = B^-_\ell$. D^+_v is the maximum length of a simple path from v to ℓ. We assume that b_j aborts when it detects any deviation from the protocol.

In the asset claiming phase, the main leader ℓ is the first to claim the incoming contracts. The behavior of followers is the same as in Protocol BDP: they claim the incoming assets one step after all their outgoing assets were claimed. The behavior of all non-main leaders is more subtle. Each such sub-leader b_j waits until all its outgoing assets *in the parent component* are claimed, and then it claims all of its incoming assets.

The full protocol for non-main leaders b_j is given in Figure 3. Figure 4 shows timeout values for the reuniclus graph in Figure 2. In what follows we explain some notations used in the protocol.

As before, we use notation D^+_y for the maximum distance from y to ℓ in G. We also need the concept analogous to the maximum distance from a leader, but this one is a little more subtle than for bottleneck graphs, because we now need to consider paths whose initial bottleneck vertex can be repeated once on the path. Formally, if $v \in G_i$, then B^-_v denotes the maximum length of a path with the following properties: (i) it starts at some leader b_j that is a descendant of b_i (possibly $b_j = b_i$), (ii) it ends in v, and (iii) it does not repeat any vertices, with one possible exception: it can only revisit b_j, and if it does, it either ends or leaves G_j (and continues in the parent component of b_j). (This can be interpreted as a maximum path length in a DAG obtained by splitting each leader into two vertices, one with the outgoing arcs into its home component and the other with all other arcs.) For example, in the graph in Figure 2, one allowed path for $v = u$ is $x - y - z - d - g - x - j - u$.

These values can be computed using auxiliary values B^-_{uv} defined for each edge (u, v). Call an edge (u, v) a *bottleneck edge* if u is a bottleneck vertex, say $u = b_j$, and $v \in G_j$. That is, bottleneck edges are the edges from bottleneck vertices that go into their home components. First, for each bottleneck edge (b_j, v) let $B^-_{b_j v} = 0$. Then, for each vertex u and each non-bottleneck edge (u, v) let $B^-_u = B^-_{uv} = \max_{(x,u)} B^-_{xu} + 1$, where the maximum is over all edges (x, u) entering u. By B^* we denote the value of B^-_ℓ.

The values B^-_z determine contract creation times. As shown in Figure 3, each leader b_j creates its contracts in its home component at time 0. Each other contract (u, v) will be created at time B^-_u. The last contract will be created by some in-neighbor of ℓ at time step $B^* - 1$. Then ℓ will initiate the contract claiming phase at time B^*. Analogous to Protocol BDP, each party u will claim its incoming contracts at time $B^* + D^+_u$, which is its timeout value.

▶ **Theorem 4.** *If G is a reuniclus digraph, then Protocol RDP is an atomic swap protocol for G.*

Proof. According to Lemma 2, it is sufficient to prove only the liveness and safety properties.

Liveness. The liveness property is quite straightforward. Each party $u \neq \ell$ has exactly one time unit, after its last incoming contract is created, to create its outgoing contracts. This will complete the contract creation at time $B^* - 1$. Thus at time B^* leader ℓ can claim

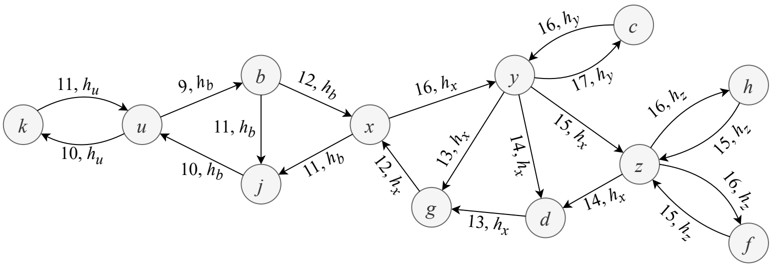


Figure 4 Timeout values and hashlocks for Protocol RDP for the graph in Figure 2. The main leader is $\ell = b$. We have $B^* = 9$ (this is the length of path $y - c - y - z - d - g - x - j - u - b$).

its incoming assets. For any other party u, each incoming asset (x, u) of u has timeout $\tau_{xu} = B^* + D_u^+$. If u is a follower then all the outgoing assets of u will be claimed before time τ_{xu}, and if u is a non-main leader then all of u's outgoing assets in its parent component will be claimed before time τ_{xu}. So u can claim all incoming assets at time τ_{xu}.

Safety. The proof of the safety condition for the main leader ℓ and pure followers is the same as in Protocol BDP for bottleneck digraphs. So here we focus only on non-main leaders.

Let b_j be a non-main leader whose parent component is G_i. Assume that b_j follows the protocol. So, according to Protocol RDP, b_j will create its outgoing contracts in G_j at time 0. Before creating its outgoing contracts in G_i, b_j checks if *all* incoming contracts are created. If any of its incoming contracts is not created or not valid, b_j will abort without creating its outgoing contracts in G_i. Thus its outcome will be NoDeal.

We can thus assume that all incoming contracts of b_j are created and correct; in particular all incoming contracts in G_j have hashlock h_j and all incoming contracts in G_i have the same hashlock h (which may or may not be equal to h_i). Then b_j creates its outgoing contracts in G_i, as in the protocol. We now need to argue that if any of b_j's outgoing assets is successfully claimed then b_j successfully claims *all* its incoming assets.

Suppose that some outgoing asset of b_j, say (b_j, w) is successfully claimed by w. Two cases arise, depending on whether w is in G_i or G_j.

If $w \in G_i$ then from contract (b_j, w) will provide b_j with some secret value s for which $H(s) = h$, because b_j used h for its outgoing contracts in G_i. At this point, b_j has both secret values s and s_j, and the timeout of all incoming contracts of b_j is greater than the timeout of (b_j, w). Therefore b_j has the correct secrets and at least one time unit to claim all incoming contracts, and its outcome will be Deal or Discount, thus acceptable.

In the second case, $w \in G_j$, the home component of b_j. For w to successfully claim (b_j, w), it must have the value of s_j. But, as b_j follows the protocol, it releases s_j only when claiming all incoming assets. So at this time b_j already has all incoming assets. Therefore in this case the outcome of b_j is also either Deal or Discount. ◀

4 A Characterization of Digraphs that Admit HTLC-Based Protocols

This section proves Theorem 1. By straightforward inspection, Protocol RDP from Section 3.2 is HTLC-based: each party creates at most one secret/hashlock pair, and all contracts are transferred using HTLC's. This already proves the (⇐) implication in Theorem 1.

It remains to prove the (⇒) implication, namely that the existence of an atomic HTLC-based protocol implies the reuniclus property of the underlying graph. We divide the proof into two parts. First, in Section 4.1 we establish some basic properties of HTLC-based protocols. Using these properties, we then wrap up the proof of the (⇒) implication in Section 4.2.

4.1 Basic Properties of HTLC-Based Protocols

Let $\mathbb{P}$ be an HTLC-based protocol for a strongly connected digraph G, and for the rest of this section assume that $\mathbb{P}$ is atomic. We now establish some fundamental properties that must be satisfied by $\mathbb{P}$.

Most of the proofs of protocol properties given below use the same fundamental approach, based on an argument by contradiction: we show that if $\mathbb{P}$ did not satisfy the given property then there would exist a (non-conforming) execution of $\mathbb{P}$ in which some parties, by deviating from $\mathbb{P}$, would force a final outcome of some conforming party to be unacceptable, thus violating the safety property.

For illustration, we include the proofs for Lemma 5 and for some other theorems and corollaries later in the section. See the full version [5] for the missing proofs.

▶ **Lemma 5.** *If some party successfully claims an incoming asset at some time t, then all contracts in the whole graph must be placed before time t.*

Proof. Assume that a party v successfully claims an asset (u, v) at time t. Towards contradiction, suppose that there is some arc (x, y) for which the contract is still not placed at time t. Since G is strongly connected, there is a path $y = u_1, u_2, ..., u_p = u$ from y to u in G. Let also $u_0 = x$ and $u_{p+1} = v$. Now consider an execution of $\mathbb{P}$ in which all parties except x are conforming, x follows $\mathbb{P}$ up to time $t - 1$, but later it never creates contract (x, y). This execution is indistinguishable from the conforming execution up until time $t - 1$, so at time t node v will claim contract (u, v). Since the first asset on path $u_0, u_1, ..., u_{p+1}$ is not transferred and the last one is, there will be a party u_j on this path, with $1 \leq j \leq p$, for which asset (u_{j-1}, u_j) is not transferred but (u_j, u_{j+1}) is. But then the outcome of u_j is unacceptable even though u_j is honest, contradicting the safety property of $\mathbb{P}$. ◀

Lemma 5 is important: it implies that $\mathbb{P}$ must consist of two phases: the *contract creation* phase, in which all parties place their outgoing contracts (by the liveness property, all contracts must be created), followed by the *asset claiming* phase, when the parties claim their incoming assets.

▶ **Lemma 6.** *Suppose that at a time t a party v creates an outgoing contract protected by a party different than v. Then all v's incoming contracts must be created before time t.*

Consider now the snapshot of of $\mathbb{P}$ right after the contract creation phase, when all contracts are already in place but none of the assets are yet claimed. Lemma 6 implies the following:

▶ **Corollary 7.**

(a) *If on some path each contract except possibly the first is not protected by its seller, then along this path the contract creation times strictly increase.*

(b) *Each cycle must contain a contract protected by its seller.*

Next, we establish some local properties of $\mathbb{P}$; in particular we will show that for each party v there is at most one other party that protects contracts involving v.

▶ **Lemma 8.** *If a party v has an incoming contract protected by some party x different from v then:*

(a) *Party v has at least one outgoing contract protected by x;*

(b) *All contracts involving v are protected either by v or by x.*

▶ **Lemma 9.** *Let $P = u_1, u_2, ..., u_k$ be a simple path whose last contract is protected by some party $z \notin \{u_1, u_2, ..., u_{k-1}\}$. Then for each $i = 1, ..., k-1$, contract (u_i, u_{i+1}) is protected by one of the parties $u_{i+1}, u_{i+2}, ..., u_{k-1}, z$. Consequently, each contract on P is not protected by its seller.*

▶ **Lemma 10.** *If all incoming contracts of a party v are protected by v then all outgoing contracts of v are also protected by v.*

Intuitively, if v had an outgoing contract protected by some other party x but not an incoming contract protected by x, then this outgoing contract would be "redundant" for v, since v does not need the secret from this contract to claim an incoming contracts. The lemma shows that the issue is not just redundancy – this is in fact not even possible if the protocol is atomic.

▶ **Lemma 11.** *If a party v has an outgoing contract protected by some party x different from v then it has an incoming contract protected by x.*

▶ **Lemma 12.** *A party v has an incoming contract protected by v if and only if it has an outgoing contract protected by v.*

The theorem below summarizes the local properties of the contracts involving a party v.

▶ **Theorem 13.** *Consider the contracts involving a party v, both incoming and outgoing.*

(a) *For each party x (which may or may not be v), v has an incoming contract protected by x if and only if v has an outgoing contract protected by x.*

(b) *If there are any contracts protected by v, then at least one incoming contract protected by v has a smaller timeout than all outgoing contracts protected by v.*

(c) *There is at most one party $x \neq v$ that protects a contract involving v. For this x, all timeouts of the outgoing contracts protected by x are smaller than all timeouts of the incoming contracts (no matter what party protects them).*

Proof.

(a) This part is just a restatement of the properties established earlier. For $x = v$, the statement is the same as in Lemma 12. For $x \neq v$, if v has an incoming contract protected by x then, by Lemma 8, it must have an outgoing contract protected by x, and if v has an outgoing contract protected by x then, by Lemma 11, it must have an incoming contract protected by x.

(b) Let (v, w) be an outgoing contract protected by v whose timeout value τ_{vw} is smallest. Consider any path P from w to v with all contracts on P protected by v. (This path must exist. To see this, starting from w follow contracts protected by v. By Corollary 7(b), eventually this process must end at v.) Then part (b) of Theorem 13 implies that along this path timeout values must decrease, so its last contract (u, v) must satisfy $\tau_{uv} < \tau_{vw}$. Thus, by the choice of w, the timeout value of (u, v) is smaller than all timeout values of the outgoing contracts protected by v.

(c) Let $x \neq v$. If v has an incoming contract protected by x then, by Lemma 8, all contracts involving v but not protected by v are protected by x. If v has an outgoing contract protected by x then Lemma 11 implies that some incoming contract is protected by x.

We now consider the claims about the timeout values. Let (v, w) be an outgoing contract protected by x, and let (u, v) be an incoming contract. Towards contradiction, suppose that in $\mathbb{P}$ the timeouts of these contracts satisfy $\tau_{uv} \leq \tau_{vw}$. Denote by t^* the first step of $\mathbb{P}$ after the contract creation phase. We consider two cases, depending on whether (u, v) is protected by x or v.

Case 1: contract (u, v) is protected by x. We consider an execution of $\mathbb{P}$ where all parties follow $\mathbb{P}$ until time $t^* - 1$. Then, starting at time t^*, we alter the behavior of some parties, as follows: all parties other than v, w and x will abort the protocol, x will provide its secret s_x to w, and w will claim asset (v, w) at time τ_{vw}. This way, the earliest v can claim asset (u, v) is at time $\tau_{vw} + 1$, which is after timeout τ_{uv}. Thus v ends up in an unacceptable outcome, giving us a contradiction, which completes the proof of (b).

Case 2: contract (u, v) is protected by v. We can assume that its timeout τ_{uv} is minimum among all incoming contracts protected by v. (Otherwise, in the argument below replace (u, v) by the incoming contract protected by v that has minimum timeout.) Let (v, y) be any outgoing contract protected by v. From part (b), we have that $\tau_{uv} < \tau_{vy}$. Let also (z, v) be any incoming contract protected by x.

We now consider an execution of $\mathbb{P}$ where all parties follow the protocol until time $t^* - 1$. At time t, all parties other than u, w, x, y, z abort the protocol, and x gives its secret s_x to w. As time proceeds, v may notice that some parties do not follow the protocol, so, even though v is honest, from this time on it is not required to follow the protocol. We show that independently of v's behavior, it can end up in an unacceptable outcome, contradicting the safety property of $\mathbb{P}$.

To this end, we consider two possibilities. If v does not claim (u, v) at or before time τ_{uv}, then w can claim (v, w) at time τ_{vw}, so v will lose asset (v, w) without getting asset (u, v). On the other hand, if v claims (u, v), then u can give secret s_v to y who can then claim asset (v, y), and w will not claim asset (v, w), so v will not be able to claim asset (z, v), as it will not have secret s_x. In both cases, the outcome of v is unacceptable. ◀

We now use the above properties to establish some global properties of $\mathbb{P}$. The first corollary extends Corollary 7, and is a direct consequence of Theorem 13(b).

▶ **Corollary 14.** *If on some path each contract except possibly the first is not protected by the seller, then along this path the timeout values strictly decrease.*

The next corollary follows from Corollaries 7 and 14.

▶ **Corollary 15.** *Let P be a path such that all contracts on P except possibly the first are protected by a party x that is not an internal vertex of P. Then all contract creation times along P strictly increase and all timeout values strictly decrease.*

▶ **Corollary 16.**

(a) *Let (u, v) be a contract protected by some party x other than v. Consider a path P starting with arc (u, v), that doesn't contain x as an internal vertex and on which each contract is not protected by its seller. Then all contracts along P are protected by x.*

(b) *Let (u, v) be a contract protected by some party x other than u. Consider a path P ending with arc (u, v), that doesn't contain x as an internal vertex and on which each contract is not protected by its buyer. Then all contracts along P are protected by x.*

Proof.

(a) The corollary follows easily by repeated application of Theorem 13. Let (v, w) be the second arc on P. By the assumption, (v, w) is not protected by v, and since v has an incoming contract protected by x and $x \neq v$, Theorem 13 implies that contract (v, w) must be also protected by x. If $w = x$, this must be the end of P. If $w \neq x$, then w has an incoming contract protected by x, so we can repeat the same argument for w, and so on. This implies part (a).

(b) The proof for this part is symmetric to that for part (a), with the only difference being that we proceed now backwards from u along P. ◀

▶ **Theorem 17.**

(a) *Let P be a simple path starting at a vertex x whose last contract is protected by x. Then all contracts on P are protected by x.*

(b) *Let Q be a simple path ending at a vertex x whose first contract is protected by x. Then all contracts on Q are protected by x.*

Proof. (a) Let $P = u_1, u_2, ..., u_{p+1}$, where $u_1 = x$ and (u_p, u_{p+1}) is protected by x. The proof is by contradiction. Suppose that P violates part (a), namely it contains a contract not protected by x. (In particular, this means that $p \geq 2$.) We can assume that among all simple paths that violate property (a), P is shortest. (Otherwise replace P in the argument below by a shortest violating path.) Then (u_{p-1}, u_p) is not protected by x, because otherwise the prefix of P from x to u_p would be a violating path shorter than P. So (u_{p-1}, u_p) is protected by u_p. Using Lemma 9, each contract on the path $u_1, u_2, ..., u_p$ is not protected by the seller. Since (u_p, u_{p+1}) is protected by x and $x \neq u_p$, each contract on P is not protected by its seller.

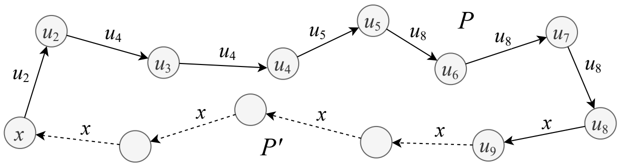


Figure 5 Illustration of the proof of Theorem 17(a). Path P is marked with solid arrows, path P' is marked with dashed arrows. Here, $p = 8$ and $u_1 = x$. The labels on edges show the parties that protect them.

Next, we claim that there is a simple path P' from u_{p+1} to x whose all contracts are protected by x. If $u_{p+1} = x$, this is trivial, so assume that $u_{p+1} \neq x$. Then, since u_{p+1} has an incoming contract protected by x and $x \neq u_{p+1}$, u_{p+1} must have an outgoing contract (u_{p+1}, w) protected by x. If $w = x$, we are done. Else, we repeat the process for w, and so on. Eventually, extending this path we must end at x, for otherwise we would have a cycle with all contracts protected by x but not containing x, contradicting Corollary 7(b). This proves that such path P' exists. Since all contracts on P' are protected by x, they are not protected by their sellers.

Finally, let C be the cycle obtained by combining paths P and P'. (See Figure 5.) Then every contract on C is not protected by its seller, contradicting Corollary 7, completing the proof. ◀

4.2 Proof of the (⇐) Implication in Theorem 1

In this section we use the protocol properties established in the previous section to prove necessary conditions for a digraph to admit an atomic HTLC protocol. We start with protocols that use only one common hashlock for the whole graph. (See the full version for the missing proofs.)

▶ **Lemma 18.** *Suppose that G has an atomic HTLC protocol $\mathbb{P}$ in which only one party creates a secret/hashlock pair. Then G must be a bottleneck graph.*

We now consider the general case, when all parties are allowed to create secret/hashlock pairs. The lemma below establishes the (⇒) implication in Theorem 1.

▶ **Lemma 19.** *Suppose that G has an atomic HTLC protocol $\mathbb{P}$. Then G must be a reuniclus digraph.*

Proof. Recall what we have established so far in Section 4.1. From Theorem 13(c) we know that, for each party u, all contracts involving u are protected either by u or by just one other party. Using this property, we define the control relation on parties, as follows: If u has any contracts protected by some other party x, we will say that x *controls* u. Let $\mathcal{K}$ be a digraph whose vertices are the parties that created secret/hashlock pairs, and each arc (u, x) represents the control relation, meaning that x controls u. We want to prove that $\mathcal{K}$ is a tree.

Each node in $\mathcal{K}$ has at most one outgoing arc. This property already implies that each connected component of $\mathcal{K}$ is a so-called *1-tree*, namely a graph that has at most one cycle. So in order to show that $\mathcal{K}$ is actually a tree, it is sufficient to show the two claims below.

▷ Claim 20. $\mathcal{K}$ does not have any cycles.

We now prove Claim 20. Towards contradiction, suppose that $\mathcal{K}$ has a cycle $C = v_1, v_2, ..., v_k, v_{k+1}$, where $v_{k+1} = v_1$. Consider any arc (v_i, v_{i+1}) on C. This arc represents that v_{i+1} controls v_i. So, in G, v_i has an outgoing contract protected by v_{i+1}. Let P_i be any path in G starting with this contract and ending at v_{i+1}. Then Theorem 17(b) gives us that all contracts on P_i are protected by v_{i+1}. Combining these paths $P_1, ..., P_k$ we obtain a cycle C' in G. Then in C', each contract is not protected by its seller, which would contract Corollary 7(b). This completes the proof of Claim 20.

▷ Claim 21. $\mathcal{K}$ has only one tree.

From Claim 20 we know that each (weakly) connected component of $\mathcal{K}$ is a tree. The roots of these trees have the property that their contracts are not protected by any other party. To prove Claim 21, suppose towards contradiction that $\mathcal{K}$ has two different trees, and denote by r and r' the roots of these trees. Since r, r' are roots of trees, all contracts involving r are protected by r and all contracts involving r' are protected by r'. Consider any simple path $P = u_1, u_2, ..., u_k$ from $u_1 = r$ to $r' = u_k$. Since the last contract on P is protected by r' and r' is not in $\{u_1, u_2, ..., u_{k-1}\}$, Lemma 9 implies that all contracts on this path are not protected by their sellers. But this contradicts the fact that (u_1, u_2) is protected by u_1. This completes the proof of Claim 21.

We now continue with the proof of the theorem. Denote by $b_1, b_2, ..., b_p$ the nodes of $\mathcal{K}$. For each b_j, define G_j to be the subgraph induced by the contracts protected by b_j. That is, for each contract (u, v) protected by b_j we add vertices u, v and arc (u, v) to G_j. The necessary properties (rg1) and (rg2) follow from our results in Section 4.1. It remains to show that subgraphs $G_1, G_2, ..., G_p$, together with tree $\mathcal{K}$, satisfy conditions (rg1) and (rg2) that characterize reuniclus graphs.

Consider some $u \neq b_j$ that is in G_j. By the definition of G_j, u is involved in a contract protected by b_j. Then Theorem 13 gives us that u has both an incoming and outgoing contract protected by b_j. Take any path P starting at an outgoing contract of u protected by b_j and ending at b_j. Then Theorem 17(b) implies that all contracts on P are protected by b_j. By the same reasoning, there is a path P' starting at b_j and ending with an incoming contract of u protected by b_j. Then, by Theorem 17(a) all contracts on P' are protected by b_j. This gives us that G_j is strongly connected. And, by Corollary 7, G_j cannot contain a cycle not including b_j. Therefore G_j is a bottleneck graph with b_j as its bottleneck.

We also need to prove that G_j is in fact an induced subgraph, that is, if $u, v \in G_j$ and G has an arc (u, v), then $(u, v) \in G_j$ as well. That is, we need to prove that (u, v) is protected by b_j. Suppose, towards, contradiction, that (u, v) is protected by some $b_i \neq b_j$. Then both

u and v are involved in contracts protected by both b_i and b_j, and this implies that $u = b_i$ and $v = b_j$, or vice versa. And this further implies that b_i would be protected by b_j and vice versa, which would be a cycle in $\mathcal{K}$, contradicting that $\mathcal{K}$ is a tree. This completes the proof of property (rg1).

Finally, consider property (rg2). If a vertex u is not any of designated bottleneck vertices $b_1, b_2, ..., b_p$, then, by Theorem 13, all its contracts are protected by the same party, which means that it belongs to exactly one graph G_j. On the other hand, if $u = b_j$, then again by Theorem 13, it is involved only in contracts protected by itself and one other party, say b_i. But then it belongs only to G_j and G_i, completing the proof of (rg2). ◀

5 Final Comments

We have provided a full characterization of digraphs for which there exist atomic HTLC-based protocols for asset swaps, by proving that these digraphs are exactly the class of reuniclus graphs. Our work raises several natural open problems and leads to new research directions, including the following:

- One natural extension of HTLCs would be to allow multiple hashlocks in a contract. We can show that there are non-reuniclus graphs that have atomic protocols based on such contracts, and that there are strongly connected digraphs that don't, although at this time we do not have a full characterization of digraphs that admit such protocols.

  A further extension would be to allow different hashlocks in the same contract have different timeouts. What types of graphs can be handled with such protocols?

  More generally, are there any simple generalizations of HTLCs that lead to atomic protocols for arbitrary graphs?
- Herlihy's method requires computing longest paths in graphs in order to specify the protocol's steps on a given digraph. Are longest paths truly necessary to assure the safety property in arbitrary graphs? If so, it would be interesting to prove this, say via some sort of computational-hardness result.

References

1 N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 7–17, New York, NY, USA, 1997. ACM. doi:10.1145/266420.266426.

2 N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18:593–610, 1997.

3 M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Trans. Inf. Theor.*, 36(1):40–46, September 2006.

4 Eric Chan, Marek Chrobak, and Mohsen Lesani. Cross-chain swaps with preferences. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*, pages 261–275. IEEE, 2023. doi:10.1109/CSF57540.2023.00031.

5 Emily Clark, Chloe Georgiou, Katelyn Poon, and Marek Chrobak. On htlc-based protocols for multi-party cross-chain swaps. *CoRR*, abs/2403.03906, 2024. arXiv:2403.03906, doi:10.48550/arXiv.2403.03906.

6 Apoorvaa Deshpande and Maurice Herlihy. Privacy-preserving cross-chain atomic swaps. In *International Conference on Financial Cryptography and Data Security*, pages 540–549. Springer, 2020. doi:10.1007/978-3-030-54455-3_38.

7 Matt Franklin and Gene Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In Rafael Hirchfeld, editor, *Financial Cryptography*, pages 90–102, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

8 Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The Arwen trading protocols. In *International Conference on Financial Cryptography and Data Security*, pages 156–173. Springer, 2020. doi:10.1007/978-3-030-51280-4_10.

9 Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 245–254, New York, NY, USA, 2018. ACM. URL: https://dl.acm.org/citation.cfm?id=3212736.

10 Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *arXiv preprint arXiv:1905.09743*, 2019. arXiv:1905.09743.

11 Soichiro Imoto, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Atomic cross-chain swaps with improved space and local time complexity, 2019. arXiv:1905.09985.

12 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1916–1933, 2018. doi:10.1137/1.9781611975031.125.

13 Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/.

14 Subhra Mazumdar. Towards faster settlement in htlc-based cross-chain atomic swaps, 2022. arXiv:2211.15804, doi:10.48550/arXiv.2211.15804.

15 Silvio Micali. Simple and fast optimistic protocols for btcwiki electronic exchange. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 12–19, New York, NY, USA, 2003. ACM.

16 Wallstreetmojo Team. Hashed timelock contract. https://www.wallstreetmojo.com/hashed-timelock-contract/, 2024.

17 Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sánchez. Universal atomic swaps: Secure exchange of coins across all blockchains. *Cryptology ePrint Archive*, 2021.

18 Tier Nolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, 2013. [Online; accessed 23-January-2021].

19 Aisshwarya Tiwari. An introductory guide to hashed timelock contracts. https://crypto.news/learn/an-introductory-guide-to-hashed-timelock-contracts/, 2022.

20 Yingjie Xue and Maurice Herlihy. Hedging against sore loser attacks in cross-chain transactions. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 155–164, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467904.

Simple Realizability of Abstract Topological Graphs

Giordano Da Lozzo
Roma Tre University, Italy

Walter Didimo
University of Perugia, Italy

Fabrizio Montecchiani
University of Perugia, Italy

Miriam Münch
University of Passau, Germany

Maurizio Patrignani
Roma Tre University, Italy

Ignaz Rutter
University of Passau, Germany

Abstract

An *abstract topological graph* (*AT-graph*) is a pair $A = (G, \mathcal{X})$, where $G = (V, E)$ is a graph and $\mathcal{X} \subseteq \binom{E}{2}$ is a set of pairs of edges of G. A *realization of* A is a drawing Γ_A of G in the plane such that any two edges e_1, e_2 of G cross in Γ_A if and only if $(e_1, e_2) \in \mathcal{X}$; Γ_A is *simple* if any two edges intersect at most once (either at a common endpoint or at a proper crossing). The AT-GRAPH REALIZABILITY (ATR) problem asks whether an input AT-graph admits a realization. The version of this problem that requires a simple realization is called SIMPLE AT-GRAPH REALIZABILITY (SATR). It is a classical result that both ATR and SATR are NP-complete [16, 19].

In this paper, we study the SATR problem from a new structural perspective. More precisely, we consider the size $\lambda(A)$ of the largest connected component of the *crossing graph* of any realization of A, i.e., the graph $\mathcal{C}(A) = (E, \mathcal{X})$. This parameter represents a natural way to measure the level of interplay among edge crossings. First, we prove that SATR is NP-complete when $\lambda(A) \geq 6$. On the positive side, we give an optimal linear-time algorithm that solves SATR when $\lambda(A) \leq 3$ and returns a simple realization if one exists. Our algorithm is based on several ingredients, in particular the reduction to a new embedding problem subject to constraints that require certain pairs of edges to alternate (in the rotation system), and a sequence of transformations that exploit the interplay between alternation constraints and the SPQR-tree and PQ-tree data structures to eventually arrive at a simpler embedding problem that can be solved with standard techniques.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Human-centered computing → Graph drawings

Keywords and phrases Abstract Topological Graphs, SPQR-Trees, Synchronized PQ-Trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.23

Related Version *Full Version*: `http://arxiv.org/abs/2409.20108` [5]

Funding Research by Da Lozzo, Didimo, Montecchiani, and Patrignani was supported, in part, by MUR of Italy (PRIN Project no. 2022ME9Z78 – NextGRAAL and PRIN Project no. 2022TS4Y3N – EXPAND). Research by Münch and Rutter was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 541433306.

Acknowledgements Research started at the 1^{st} Summer Workshop on Graph Drawing, September 2021, Castiglione del Lago, Italy.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *topological graph* Γ is a geometric representation of a graph $G = (V, E)$ in the plane such that the vertices of G are mapped to distinct points and the edges of G are simple curves connecting the points corresponding to their end-vertices. For simplicity, the geometric representations of the elements of V and E in Γ are called *vertices* and *edges* of Γ, respectively. It is required that: (*i*) any intersection point of two edges in Γ is either a common endpoint or a crossing (a point where the two edges properly cross); (*ii*) any two edges of Γ have finitely many intersections and no three edges pass through the same crossing point. Additionally, we say that Γ is *simple* if adjacent edges never cross and any two non-adjacent edges cross at most once. The *crossing graph* $\mathcal{C}(\Gamma)$ of a topological graph Γ is a graph whose vertices correspond to the edges of Γ (and hence of G) and two vertices are adjacent if and only if their corresponding edges cross in Γ.

An *abstract topological graph* (*AT-graph*) is a pair $A = (G, \mathcal{X})$ such that $G = (V, E)$ is a graph and $\mathcal{X} \subseteq \binom{E}{2}$ is a set of pairs of edges of G. We say that A is *realizable* if there exists a topological graph Γ_A isomorphic to G such that any two edges e and e' of G cross (possibly multiple times) in Γ_A if and only if $(e, e') \in \mathcal{X}$. The topological graph Γ_A is called a *realization of A*. Note that, by definition, A is realizable if and only if the crossing graph of any realization Γ_A of A is isomorphic to the graph $(E, \mathcal{X})$. Since such a crossing graph only depends on A (i.e., it is the same for any realization of A), we denote it by $\mathcal{C}(A)$.

The AT-GRAPH REALIZABILITY (ATR) problem asks whether an AT-graph $A = (G, \mathcal{X})$ is realizable. The SIMPLE AT-GRAPH REALIZABILITY (SATR) problem is the version of ATR in which the realization of A is required to be simple; if such a realization exists, then A is said to be *simply realizable*. Since the introduction of the concept of AT-graphs [18], establishing the complexity of the ATR (and of the SATR) problem has been the subject of an intensive research activity, also due to its connection with other prominent problems in topological and geometric graph theory. Clearly, if $\mathcal{X} = \emptyset$, both the ATR and the SATR problems are equivalent to testing whether G is planar, which is solvable in linear time [3, 15]. However, for $\mathcal{X} \neq \emptyset$, a seminal paper by Kratochvíl [16] proves that ATR is NP-hard and that this problem is polynomially equivalent to recognizing *string graphs*. We recall that a graph S is a string graph if there exists a system of curves (called *strings*) in the plane whose crossing graph is isomorphic to S. In the same paper, Kratochvíl proves the NP-hardness of the WEAK AT-GRAPH REALIZABILITY (WATR) problem, that is, deciding whether a given AT-graph $A = (G, \mathcal{X})$ admits a realization where a pair of edges may cross only if it belongs to $\mathcal{X}$. He also proves that recognizing string graphs remains polynomial-time reducible to WATR. Subsequent results focused on establishing decision algorithms for WATR; it was first proven that this problem belongs to NEXP [25] and then to NP [24]. This also implies the NP-completeness of recognizing string graphs (which is polynomial-time reducible to WATR) and of ATR (which is polynomially equivalent to string graph recognition). Concerning the simple realizability setting for AT-graphs, it is known that the SATR problem remains NP-complete, still exploiting the connection with recognizing string graphs [19, 20]. On the positive side, for those AT-graphs $A = (G, \mathcal{X})$ for which G is a complete n-vertex graph, SATR is solvable in polynomial-time, with an $O(n^6)$-time algorithm [21, 22]. Refer to [21] for the complexity of other variants of ATR, and to [11] for a connection with the popular SIMULTANEOUS GRAPH EMBEDDING problem.

Our contributions. In this paper, we further investigate the complexity of the simple realizability setting, i.e., of the SATR problem. We remark that focusing on simple drawings is a common scenario in topological graph theory, computational geometry, and graph drawing (see, e.g., [9, 12, 14, 26]), because avoiding crossings between adjacent edges, as well as multiple crossings between a pair of non-adjacent edges, is a requirement for minimal edge crossing layouts. Specifically, we study the simple realizability problem for an AT-graph A from a new structural perspective, namely looking at the number of vertices of the largest connected component of the crossing graph $\mathcal{C}(A)$, which we denote by $\lambda(A)$. This parameter is a natural measure of the level of interplay among edge crossings. Namely, SATR is trivial on instances for which $\lambda(A) \leq 2$, that is, instances in which the number of crossings is unbounded but each edge is crossed at most once. On the other hand, the problem becomes immediately nontrivial as soon as $\lambda(A) \geq 3$. Precisely, our results are as follows:

- We prove that Simple AT-graph Realizability is NP-complete already for instances A for which $\lambda(A) = 6$ (which, in fact, implies the hardness for every fixed value of $\lambda(A) \geq 6$); see Section 3. A consequence of our result is that, unless P = NP, the problem is not fixed-parameter tractable with respect to $\lambda(A)$ and, thus, with respect to any graph parameter bounded by $\lambda(A)$, such as the maximum node degree, the treewidth or even the treedepth. As the results in [16, 19, 20], our hardness proof uses a reduction from 3-Connected Planar 3-SAT. However, the reduction in [16] does not deal with the simplicity of the realization, whereas the reduction in [19, 20] may lead to instances A for which $\lambda(A)$ is greater than six and, actually, is even not bounded by a constant.
- We prove that Simple AT-graph Realizability can be solved efficiently when $\lambda(A) \leq 3$. More precisely, we give an optimal $O(n)$-time testing algorithm, which also finds a simple realization if one exists; see Section 4. We remark that the only polynomial-time algorithm previously known in the literature for the SATR problem is restricted to complete graphs and has high complexity [21, 22]. Our algorithm is based on several ingredients, including the reduction to a new embedding problem subject to constraints that require certain pairs of edges to alternate (in the rotation system), and a sequence of transformations that exploit the interplay between alternation constraints and the SPQR-tree [7, 8] and PQ-tree [3, 4] data structures to eventually arrive at a simpler embedding problem that can be solved with standard techniques. We remark that the alternation constraints we encounter in our problem are rather opposite to the more-commonly encountered consecutivity constraints [1, 2, 13, 23] and cannot be handled straightforwardly with PQ-trees.

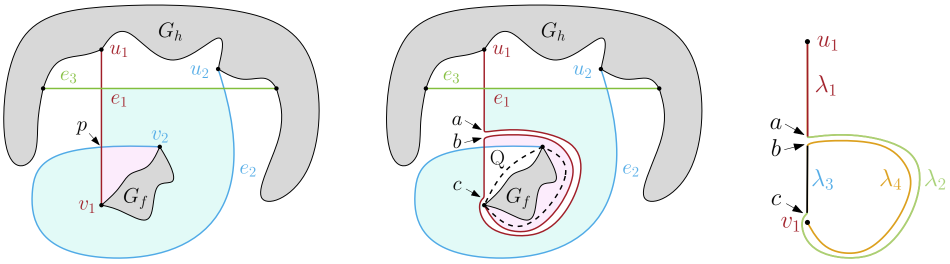
For proofs of lemmas and theorems marked with ($\star$) we refer to the full version [5].

2 Basic Definitions and Tools

For basic definitions about graphs and their drawings, refer to [6, 10]. We only consider simple realizations and thus we often omit the qualifier “simple” when clear from the context.

Let $A = (G, \mathcal{X})$ be an AT-graph, with $G = (V, E)$, and let Γ_A be a realization of A. A *face* of Γ_A is a region of the plane bounded by maximal uncrossed portions of the edges in E. A set $E' \subseteq E$ of k edges pairwise crossing in Γ_A is a *k-crossing*of Γ_A. As we focus on simple realizations, we assume that the edges in E' are pairwise non-adjacent in G. For a k-crossing E', denote by $V(E')$ the set of $2k$ endpoints of the k edges in E'. The *arrangement* of E', denoted by $C_{E'}$, is the arrangement of the curves representing the edges of E' in Γ_A. A k-crossing E' is *untangled* if, in the arrangement $C_{E'}$, all $2k$ vertices in $V(E')$ are incident to a common face (see Fig. 1b); otherwise, it is *tangled* (see Fig. 1a). The next lemma will turn useful in Section 4.

▶ **Lemma 1** ($\star$). *An AT-graph A with $\lambda(A) \leq 3$ admits a simple realization if and only if it admits a simple realization in which all 3-crossings are untangled.*



(a) Realization Γ_A. **(b)** Realization Γ'_A. **(c)** Curves of e_1.

Figure 1 Illustrations for the proof of Lemma 1. (a) A schematic representation of a simple realization Γ_A of an AT-graph A with a tangled 3-crossing $E' = \{e_1, e_2, e_3\}$. (b) The simple realization Γ'_A obtained from Γ_A, where E' is untangled. (c) The curves forming e_1 in Γ'_A.

Proof Sketch. Let A be an AT-graph with $\lambda(A) \leq 3$ and let Γ_A be a simple realization of A that contains a tangled 3-crossing E'. We show how to obtain a new simple realization Γ'_A of A that coincides with Γ_A except for the drawing of one of the edges in E' and such that E' is untangled (refer to Fig. 1). Repeating such a transformation for each tangled 3-crossing yields the desired simple realization of A with no tangled 3-crossings.

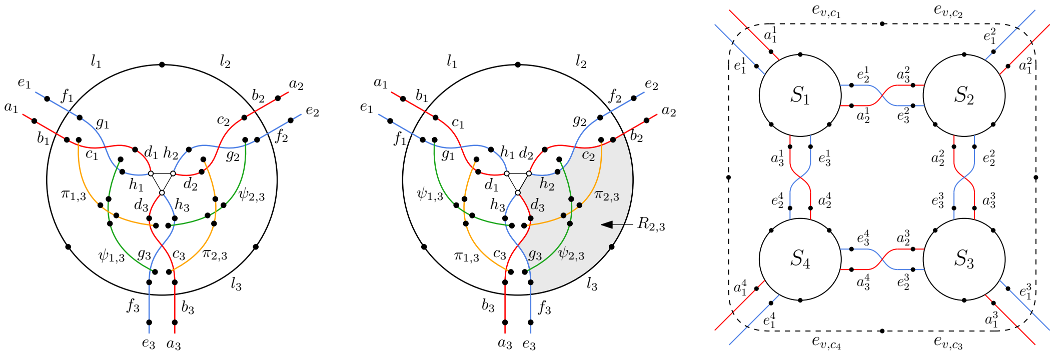
Since Γ_A is simple and $|E'| = 3$, the arrangement $C_{E'}$ in Γ_A splits the plane into two faces, a bounded face f and an unbounded face h. Let $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$, and $e_3 = (u_3, v_3)$ be the edges in E'. Since E' is tangled, assume w.l.o.g. that f contains an endpoint of two edges of E' (and thus h contains the remaining four endpoints), that such endpoints are v_1 and v_2, and that traversing e_2 from u_2 to v_2, we see u_1 to the left at the intersection between e_2 and e_1. Let G_f (resp. G_h) be the subgraph of G formed by the vertices and edges of G in the interior of f (resp. of h). Let Q be a closed curve passing through v_1 and v_2 that encloses the drawing of G_f in Γ_A, without intersecting any other vertex or edge. To obtain Γ'_A, replace the drawing of e_1 in Γ_A with the union of four curves $\lambda_1, \lambda_3, \lambda_3, \lambda_4$ defined as shown in Fig. 1c by following the drawing of e_1, e_2 and Q. Moving on e_2 from u_2 to v_2, we now see u_1 to the right at the intersection of e_2 with e_1. Hence, all the endpoints of the edges in E' lie in the same face of $C_{E'}$ in Γ'_A, i.e., E' is untangled in Γ'_A. ◀

3 NP-completeness for AT-Graphs with $\lambda(A) \geq 6$

In this section, we show that the SATR problem is NP-complete for an AT-graph A even when the largest component of the crossing graph $\mathcal{C}(A)$ has bounded size; specifically, when $\lambda(A) = 6$ (see Theorem 6). We will exploit a reduction from the NP-complete problem 3-Connected Planar 3-SAT [17].

Let φ be a Boolean formula in conjunctive normal form. The *variable-clause graph G_φ of φ* is the bipartite (multi-)graph that has a node for each variable and for each clause, and an edge between a variable-node and a clause-node if a positive or negated literal of the variable appears in the clause. If each clause of φ has exactly three literals corresponding to different variables and G_φ is planar and triconnected, then φ is an instance of 3-Connected Planar 3-SAT. Observe that in this case G_φ is a simple graph. Our proof exploits several gadgets described hereafter, which are then combined to obtain the desired reduction.

Intuitively, in the instance A_φ of SATR corresponding to φ, we encode truth values into the clockwise or counter-clockwise order in which some edges cross suitable cycles of the subgraphs (called "variable gadgets") representing the variables of φ in A_φ. These edges connect the variable gadgets to the subgraphs (called "clause gadgets") representing those clauses that contain a literal of the corresponding variable. Only if at least one of the



(a) **(b)** **(c)**

Figure 2 (a,b) The split gadget S : The clockwise circular order of the edges leaving the gadget is either b_1, f_1, b_2, f_2, b_3, f_3 (a) or f_1, b_1, f_2, b_2, f_3, b_3 (b). (c) The variable gadget $\mathcal{V}_v$. The dashed edges belong to the variable cycle of v in the skeleton H_φ.

literals appearing in a clause gadget encodes a `true` value, the clause gadget admits a simple realization. We start by describing the "skeleton" of A_φ, that is the part of A_φ that encloses all the gadgets and ensures that they are properly connected. Next, we describe the "split gadget" which, in turn, is used to construct the variable gadget. If a variable v has literals in k clauses, we have k pairs of edges leaving the corresponding variable gadget and entering the k clause gadgets. The clause gadgets always receive three truth values, corresponding to the three literals of the corresponding clause.

Skeleton. Arbitrarily choose a planar embedding $\mathcal{E}_\varphi$ of G_φ. The *skeleton* H_φ *of* φ is a 4-regular 3-connected plane graph obtained from $\mathcal{E}_\varphi$ as follows. For each degree-k variable v of φ, the graph H_φ contains a k-cycle formed by the sequence of edges $(e_{v,c_1}, e_{v,c_2}, \dots, e_{v,c_k})$, which we refer to as the *variable cycle* of v, where $c_1, \dots, c_k$ are the clause-nodes of G_φ adjacent to v in the clockwise order in which they appear around v in $\mathcal{E}_\varphi$.
For each clause c of φ, the graph H_φ contains a 3-cycle formed by the sequence of edges $(e_{c,v_1}, e_{c,v_2}, e_{c,v_3})$, which we refer to as the *clause cycle* of c, where v_1, v_2, v_3 are the variable-nodes of G_φ adjacent to c in the clockwise order in which they appear around c in $\mathcal{E}_\varphi$.
For each edge (v_i, c_j) of G_φ, the graph H_φ contains two edges, which we refer to as the *pipe edges* of the edge (v_i, c_j), connecting the endpoints of e_{v_i,c_j} and e_{c_j,v_i} without crossings.

▶ **Lemma 2** ($\star$). *The skeleton H_φ obtained from $\mathcal{E}_\varphi$ is triconnected.*

Split gadget. The split gadget S is the AT-graph defined as follows; refer to Figs. 2a and 2b. The underlying graph of S consists of six connected components: (1) a 3-cycle formed by the edges l_1, l_2, and l_3, which we call *outer cycle* of S; (2) a 3-cycle formed by the vertices v_1, v_2, and v_3 (filled white in Figs. 2a and 2b) such that, for $i = 1, 2, 3$, the vertex v_i is the endpoint of the two paths formed by the sequence of edges (a_i, b_i, c_i, d_i) (red paths in Figs. 2a and 2b) and (e_i, f_i, g_i, h_i) (blue paths in Figs. 2a and 2b); (3) four length-3 paths $\pi_{1,3}$, $\pi_{2,3}$, $\psi_{1,3}$, and $\psi_{2,3}$. We denote the first, intermediate, and last edge of a length-3 path p as p', p'', and p''', respectively. The crossing graph $\mathcal{C}(S)$ of S consists of several connected components. Next, we describe the eight *non-trivial connected components* of $\mathcal{C}(S)$, i.e., those that are not isolated vertices, determined by the following crossings of the edges of S: (i) For $j = 1, 2, 3$,

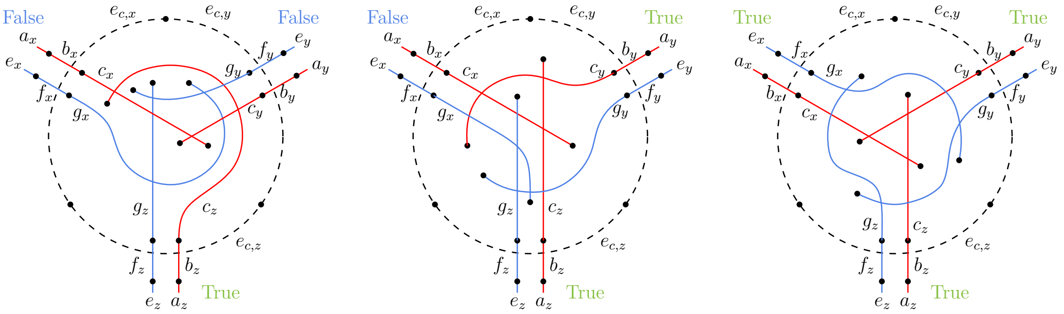


Figure 3 Illustrations for the existence of simple realizations of the clause gadget $\mathcal{Q}_c$ together with the clause cycle of c (dashed edges) when for at least one pair f_v, b_v, with $v \in \{x, y, z\}$, we have that b_v precedes f_v along $e_{c,v}$, while traversing the clause cycle of c clockwise.

edge l_j crosses both b_j and f_j; (ii) For $j = 1, 2$, edge $\pi''_{j,3}$ crosses $\psi''_{j,3}$; (iii) For $j = 1, 2$, edge c_j crosses g_j and $\pi'_{j,3}$, further g_j crosses $\psi'_{j,3}$; finally (iv) edge c_3 crosses g_3, $\pi'''_{1,3}$, and $\pi'''_{2,3}$, further g_3 crosses $\psi'''_{1,3}$ and $\psi'''_{2,3}$.

▶ **Lemma 3** ($\star$). *In any simple realization of S, the circular (clockwise or counterclockwise) order of the edges crossing the outer cycle of S is either b_1, f_1, b_2, f_2, b_3, f_3 or f_1, b_1, f_2, b_2, f_3, b_3.*

Variable gadget. For each variable v of degree k in φ and incident to clauses $c_1, c_2, \dots, c_k$ in G_φ, the *variable gadget* $\mathcal{V}_v$ is an AT-graph defined as follows; refer to Fig. 2c. Assume, w.l.o.g., that $c_1, c_2, \dots, c_k$ appear in this clockwise order around v in $\mathcal{E}_\varphi$. The underlying graph of $\mathcal{V}_v$ is composed of k split gadgets $S_1, S_2, \dots, S_k$. For each split gadget S_i, with $i = 1, \dots, k$, rename the edges a_j and e_j of S_i as a^i_j and e^i_j, respectively, with $j \in \{1, 2, 3\}$. For $i = 1, \dots, k$, we identify the edges a^i_j and a^{i+1}_{j+1} and the edges e^i_j and e^{i+1}_{j+1}, where $k + 1 = 1$. The crossing graph $\mathcal{C}(\mathcal{V}_v)$ of $\mathcal{V}_v$ consists of all vertices and edges of the crossing graphs of S_i, with $i = 1, \dots, k$. Moreover, for $i = 1, \dots, k$, it contains a non-trivial connected component consisting of the single edge (a^i_2, e^i_2) (which coincides with (a^{i+1}_3, e^{i+1}_3), $i + 1 = 1$ when $i = k$).

▶ **Lemma 4** ($\star$). *In any simple realization of $\mathcal{V}_v$ together with the variable cycle of v in which both a^i_1 and e^i_1 cross e_{v,c_i}, for $i = 1, \dots, k$, the clockwise circular order of the edges crossing the variable cycle of v in $\mathcal{V}_v$ is either a^1_1, e^1_1, a^2_1, $e^2_1, \dots,$ a^k_1, e^k_1 or e^1_1, a^1_1, e^2_1, $a^2_1, \dots,$ e^k_1, a^k_1.*

In the proof of Theorem 6, the two circular orders for the edges $\bigcup_{i=1}^k \{a^i_1, e^i_1\}$ of $\mathcal{V}_v$ considered in Lemma 4 will correspond to the two possible truth assignment of the variable v.

Clause gadget. For each clause c in φ, the *clause gadget* $\mathcal{Q}_c$ is the AT-graph, whose construction is inspired by a similar gadget used in [19], defined as follows; see Fig. 3. The underlying graph of $\mathcal{Q}_c$ consists of six length-3 paths: For $v \in \{x, y, z\}$, we have a path formed by the edges (a_v, b_v, c_v) (red paths in Fig. 3) and a path formed by the edges (e_v, f_v, g_v) (blue paths in Fig. 3). The crossing graph $\mathcal{C}(\mathcal{Q}_c)$ of $\mathcal{Q}_c$ consists of one non-trivial connected component formed by the triangles (c_x, c_y, c_z) and (g_x, g_y, g_z), and the edges (c_x, g_z), (c_y, g_x), and (c_z, g_y).

▶ **Lemma 5** ($\star$). *The clause gadget $\mathcal{Q}_c$ admits a simple realization together with the clause cycle of c in which, for $v \in \{x, y, z\}$, both b_v and f_v cross $e_{c,v}$, and in which the edges $e_{c,x}$, $e_{c,y}$, and $e_{c,z}$ appear in this order when traversing clockwise the clause cycle of c if and only if for at least one pair f_v, b_v, with $v \in \{x, y, z\}$, we have that b_v precedes f_v along $e_{c,v}$ when traversing the clause cycle of c clockwise.*

Based on Lemma 5, we associate the `True` value with a literal of a variable $v \in \{x, y, z\}$ appearing in c when b_v precedes f_v along $e_{c,v}$ while traversing the clause cycle of c clockwise, and `False` otherwise; see Fig. 3. We can finally prove the main result of the section.

▶ **Theorem 6** (⋆). *SATR is NP-complete for instances A with $\lambda(A) = 6$.*

Proof Sketch. The membership in NP is obvious. We give a reduction from the NP-complete problem 3-Connected Planar 3-SAT [17]. Let φ be an instance of 3-Connected Planar 3-SAT. We construct an instance $A_\varphi = (G', \mathcal{X}')$ of SATR that is simply realizable if and only if φ is satisfiable. We initialize $G' = H_\varphi$ and $\mathcal{X}' = \emptyset$. Then, for each variable v, we extend A_φ to include $\mathcal{V}_v$ as follows: For each clause c_i that contains a literal of v, add to $\mathcal{X}'$ the pair of edges $\{a^i_1, e_{v,c_i}\}$ and $\{e^i_1, e_{v,c_i}\}$, where a^i_1 and e^i_1 belong to $\mathcal{V}_v$, and e_{v,c_i} belongs to H_φ. Also, for each clause c, we extend A_φ to include $\mathcal{Q}_c$ as follows: For each variable $v \in \{x, y, z\}$ whose literals belong to c, we add to $\mathcal{X}'$ the pair of edges $\{f_v, e_{c,v}\}$ and $\{b_v, e_{c,v}\}$, where f_v and b_v belong to $\mathcal{Q}_c$, and $e_{c,v}$ belongs to H_φ. Finally, for each occurrence of a literal of a variable v to a clause c_i, we identify edges of $\mathcal{V}_v$ with edges of $\mathcal{Q}_v$ as follows: If v appears as a positive (resp. negated) literal in c_i, then we identify the edge a^i_1 of $\mathcal{V}_v$ with the edge a_y (resp. e_y) of $\mathcal{Q}_v$ and we identify the edge e^i_1 of $\mathcal{V}_v$ with the edge e_y (resp. a_y) of $\mathcal{Q}_v$. Observe that we do not allow the edges a^i_1 and e^i_1 to cross. Clearly, A_φ can be constructed in polynomial time. The equivalence between A_φ and φ immediately follows from Lemmata 4 and 5, and from the fact that, by Lemma 2, in any simple realization of A_φ, all the variable cycles and all the clause cycles maintain the same circular orientation. Finally, note that the size of the largest connected component of $\mathcal{C}(A_\varphi)$ is six. ◀

We remark that the NP-hardness of Theorem 6 holds for instances whose crossing graph is planar, and has maximum degree 3 and treewidth 3. Moreover, it implies that SATR is NP-complete when $\lambda(A) \geq k$, for any $k \geq 6$. Finally, since our reduction yields instances whose size is linear in the size of the input (planar) 3-SAT formula, we have the following.

▶ **Corollary 7** (⋆). *Unless ETH fails, SATR has no $2^{o(\sqrt{n})}$-time algorithm, where n is the number of vertices of the input AT-graph.*

4 A Linear-Time Algorithm for AT-Graphs with $\lambda(A) \leq 3$

In this section we show that the problem SATR can be solved in linear-time for AT-graphs A with $\lambda(A) \leq$ 3; see Theorem 13. We first give a short high-level overview of the overall strategy but note that proper definitions will only be given later in the detailed description of the algorithm. The first step is to reduce SATR to a constrained embedding problem where each vertex v may be equipped with alternation constraints that restrict the allowed orders of its incident edges around v. Next, we further reduce to the biconnected variant of the embedding problem which leads to new types of alternation constraints. It will turn out that many of these constraints can be transformed into constraints that can be expressed in terms of PQ-trees and are therefore easier to handle. Finally, we show that, when no further such transformations are possible, all the remaining alternation constraints have a simple structure that allows for an efficient test.

We now start with reducing SATR to a constrained embedding problem. Let $A = (G, \mathcal{X})$ be an n-vertex AT-graph such that $\lambda(A) \leq 3$. We construct from G an auxiliary graph H as follows. For each connected component X of $\mathcal{C}(A)$ that is not an isolated vertex, denote by $E(X)$ the set of edges of G corresponding to the vertices of X, and by $V(X)$ the vertices of G that are end-vertices of the edges in $E(X)$. Remove from G the edges in $E(X)$

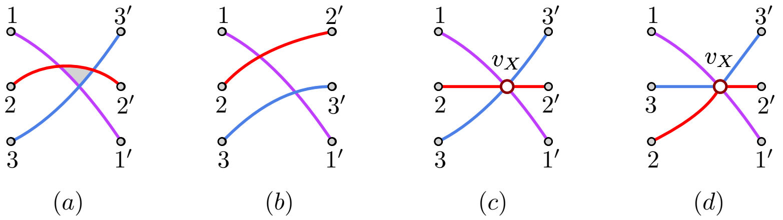


Figure 4 (a) A K_3-crossing. (b) A P_3-crossing. A circular order of the neighbors around a crossing vertex v_X satisfying (c) a K_3-constraint but not a P_3-constraint, (d) a P_3- but not a K_3-constraint.

and add a *crossing vertex* v_X adjacent to all vertices in $V(X)$; see Fig. 4. Since no two crossing vertices are adjacent, the graph H does not depend on the order in which we apply these operations. The edges incident to v_X are partitioned into pairs of edges where two edges (a, v_X) and (b, v_X) form a pair if (a, b) is an edge of G corresponding to a vertex of X. We call (a, v_X) and (b, v_X) the *portions* of (a, b) and say that (a, v_X) and (b, v_X) *stem from* (a, b) . Note that since $\lambda(A) \leq 3$, a crossing vertex v_X has either degree 4 or 6. In the first case X is a K_2 and in the latter case X is either an induced 3-path P_3 or a triangle K_3. If $X = K_2$, we color its two vertices red and blue, respectively. If $X = K_3$ we color its three vertices red, blue, and purple, respectively. If $X = P_3$, its vertex of degree 2 is colored purple, whereas we color red and blue the remaining two vertices, respectively. Based on Lemma 1, we observe the following.

▶ **Observation 8** ($\star$). *If A admits a simple realization, then H is planar.*

Observation 8 gives an immediate necessary condition for the realizability of A, which is, however, not sufficient. Indeed, a planar embedding of H obtained from contracting edges in a realization of A satisfies an additional property: for each crossing vertex v_X the portions stemming from two distinct edges e, f of G alternate around v_X *if and only* if the two vertices corresponding to e, f in X are adjacent. To keep track of this requirement, we equip every crossing vertex v_X with an *alternation constraint* that (i) colors its incident edges with colors r(ed), b(lue), p(urple) so that a portion of an edge in G gets the same color as the corresponding vertex in X, and (ii) specifies which pairs of colors must alternate around v; see Fig. 4 for an example. For a *K_2-constraint* there are no purple edges, and red and blue must alternate. For a *K_3-constraint* all pairs of colors must alternate. For a *P_3-constraint*, red and purple as well as purple and blue must alternate, whereas red and blue must not alternate. Each component X of $\mathcal{C}(A)$ with the coloring described above naturally translates to a constraint for v_X. For $X = K_2$, we obtain a K_2-constraint, for $X = P_3$ we get a P_3-constraint, and for $X = K_3$ we get a K_3-constraint; see Fig. 4. The auxiliary graph H with alternation constraints is *feasible* if it admits a planar embedding that satisfies the alternation constraints of all vertices. Thus, we have the following.

▶ **Lemma 9.** *An AT-graph $A = (G, \mathcal{X})$ with $\lambda(A) \leq 3$ is simply realizable if and only if the corresponding auxiliary graph H with alternation constraints is feasible.*

To find such an embedding, we decompose the graph into biconnected components. It turns out that this may create additional types of alternation constraints that stem from the constraints described above, but do not fall into the category of an existing class of constraints. For the sake of exposition, we introduce these constraints now, even though they will not be part of an instance obtained by the above reduction from SATR.

Let v be a vertex of degree 5 and let c be a color. For a C-constraint ($C \in \{K_3, P_3, K_2\}$) as defined above, we define a corresponding *C^{-c}-constraint* of v, which (i) colors the edges incident to v such that each color occurs at most twice but color c occurs only once and

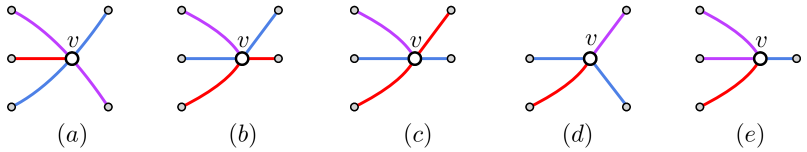


Figure 5 Circular orders of edges incident to a vertex v satisfying (a) a K_3^{-r}- and a P_3^{-r}-constraint, (b) a P_3^{-p}- but not a K_3^{-p}-constraint, (c) a K_3^{-p}- but not a P_3^{-p}-constraint,(d) a $P_3^{-(p,r)}$-but not a $K_3^{-(p,r)}$-constraint , (e) a $P_3^{-(b,r)}$- and a $K_3^{-(b,r)}$-constraint.

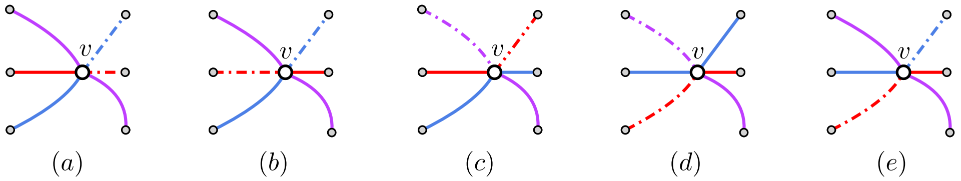


Figure 6 Circular orders of edges around a vertex v allowing to insert two edges of distinct colors (dashed) so that every color occurs twice and a $(a–b)$ K_3-constraint, $(c–e)$ P_3-constraint is satisfied.

(ii) requires that in the rotation, it is possible to insert an edge of color c so that the original C-constraint is satisfied; see Fig. 5. Observe that a K_2^{-c}-constraint is always satisfied and is thus not needed. Since the colors of a K_3-constraint are entirely symmetric, we may assume without loss of generality that $c = r$ in this case. For P_3-constraints, only red and blue are symmetric, i.e., we may assume without loss of generality that either $c = p$ or $c = r$. In particular, the K_3^{-r}-constraint and the P_3^{-r}-constraint both require that purple and blue alternate around v, whereas the position of the red edge is arbitrary; see Fig. 5(a). Thus the K_3^{-r}-constraint and the P_3^{-r}-constraint are equivalent. For a P_3^{-p}-constraint to be fulfilled, red and blue must not alternate and the purple edge either has to be between the two red edges or between the two blue edges; see Fig. 5(b).

Now let v be a vertex of degree 4 and let c, c' be two colors. For a C-constraint, we define a corresponding $C^{-c,c'}$*-constraint* of v, which (i) colors the edges incident to v such that the colors distinct from c and c' occur twice but colors c and c' occur only once if $c \neq c'$, or not at all if $c = c'$, and (ii) requires that in the rotation, it is possible to insert two edges of color c and c', respectively, so that the original C-constraint is satisfied. Since the colors of a K_3-constraint are entirely symmetric, we may assume w.l.o.g. that either $c = c' = r$ or $c = r, c' = b$ in this case. For P_3-constraints, only red and blue are symmetric, we may hence assume without loss of generality that $(c, c') \in \{(r, r), (p, p), (r, p), (r, b)\}$. Observe that a $K_2^{-c,c'}$-constraint is always satisfied and is thus not needed. The same holds for a $K_3^{-r,b}$-constraint, a $P_3^{-r,b}$-constraint and a $P_3^{-r,p}$-constraint; see Fig. 6. Also, note that a $K_3^{-r,r}$-constraint and a $P_3^{-r,r}$-constraint are both equivalent to a K_2-constraint, while a $P_3^{-p,p}$-constraint requires that red and blue do not alternate around v.

Finally for a C-constraint, we define a corresponding $C^{-(c,c')}$*-constraint* of v, which (i) colors the edges incident to v such that the colors distinct from c and c' occur twice but colors c and c' occur only once if $c \neq c'$, or not at all if $c = c'$, and (ii) requires that in the rotation, it is possible to insert an edge of color c and an edge of color c' *consecutively*, so that the C-constraint is satisfied; see Fig. 5(d), (e) for examples. This type of constraints is motivated as follows. Let v be a cut vertex in a graph G. The *cut components* of v in G are the subgraphs of G induced by v together with the maximal subsets of the vertices of G that are not disconnected by the removal of v. Note that the edges belonging to two different cut components cannot alternate around v without resulting in a crossing and observe that a $K_2^{-(c,c')}$-constraint with $c \neq c'$ is always satisfied and is thus not needed. Also note that a $C^{-(c,c)}$-constraint cannot be satisfied, since every C-constraint requires that every color alternates with at least one of the remaining colors. Since the colors of a K_3-constraint are entirely symmetric, we may assume without loss of generality that either $c = c' = r$ or $c = r, c' = b$ in this case. For P_3-constraints, only red and blue are symmetric, i.e., we may assume without loss of generality that $(c, c') \in \{(r, r), (p, p), (r, p), (r, b)\}$. In particular, a $K_3^{-(r,b)}$-constraint and a $P_3^{-(r,b)}$-constraint both require the consecutivity of the two purple edges and are thus equivalent; see Fig. 5(*e*). For a $P_3^{-(r,p)}$-constraint to be fulfilled, the two blue edges must not occur consecutively (see Fig. 5(*d*)); i.e., the two blue edges have to alternate with the two remaining edges. Thus a $P_3^{-(p,r)}$-constraint is equivalent to a K_2-constraint. By the above discussion we may assume that only K_3, P_3, K_3^{-r}, P_3^{-p}, K_2, $P_3^{-p,p}$ and $K_3^{-(r,b)}$ constraints occur.

The Alternation-Constrained Planarity (ACP) problem has as input a graph H with alternation constraints and asks whether H is feasible. By Lemma 9, there is a linear-time reduction from SATR with $\lambda(A) \leq 3$ to ACP. Next, we further reduce ACP to 2-connected ACP, which is the restriction of ACP to instances for which H is 2-connected.

▶ **Lemma 10** (⋆). *There is a linear-time algorithm that either recognizes that an instance H of* ACP *is a no-instance or computes a collection $H_1, \ldots, H_k$ of instances of* 2-connected ACP*, such that H is a yes-instance if and only if H_i is a yes-instance for every $1 \leq i \leq k$.*

Proof Sketch. Our reduction strategy considers one cut vertex at a time and splits the graph at that vertex into a collection of smaller connected components. The reduction consists of applying this cut vertex split until all cut vertices are removed or we find out that H is a no-instance. Consider an instance H of ACP and one of its cut vertices v with cut components $H_1, \ldots, H_l$. In the cut components, let every vertex except v preserve its alternation constraint (if any). Now the goal is to find out which constraints have to be assigned to the copies of v in the cut components such that H is a yes-instance if and only if each H_i is a yes-instance. We denote by $E(v)$ the edges incident to v in H and by $E_i(v)$ the edges incident to v in H_i, for $1 \leq i \leq l$. Without loss of generality assume that $|E_i(v)| \geq |E_j(v)|$ for $1 \leq i < j \leq l$. We encode the distribution of edges from $E(v)$ among the cut components as a *split-vector* $(|E_1(v)|, |E_2(v)|, \ldots, |E_l(v)|)$.

If v has no alternation constraint, H is feasible if and only if each cut component H_i, with $i = 1, \ldots, l$, is and hence all copies of v remain unconstrained. Otherwise, v has an alternation constraint C. This implies $|E(v)| \leq 6$ and thus the edges in $E(v)$ are distributed among at least two and at most six cut components. Note that the edges belonging to two different cut components cannot alternate around v without resulting in a crossing. Thus, H is a no-instance if $C \in \{K_3, K_3^{-r}, K_2\}$ and there are two cut components containing a pair of edges of the same color from $E(v)$, respectively. If $C \in \{P_3, P_3^{-p}\}$, the same holds if one cut component contains both purple edges, whereas a distinct cut component contains both red or both blue edges. In the following, we assume that the above does not apply.

Now we consider cases based on the split-vectors. If $|E_1(v)| \leq 3$, H is either a no-instance, or we can always arrange the cut components around v such that C is satisfied. In all positive cases, it suffices to leave each copy of v unconstrained. It remains to consider the remaining split-vectors with $|E_1(v)| \geq 4$. Here we only describe the case $(5, 1)$; the remaining cases can be found in the full version [5].

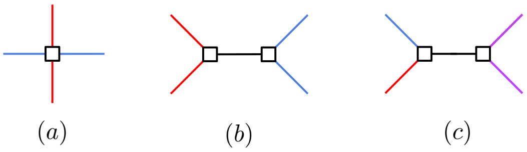


Figure 7 The PQ-trees representing alternation constraints of degree-4 vertices. (a) K_2-constraint, (b) $P_3^{-p,p}$-constraint, and (c) $K_3^{-(r,b)}$-constraint.

Case: $(5, 1)$. Let $C \in \{K_3, P_3\}$ be the constraint of v and let c be the color of the edge of $E(v)$ in H_2. To merge embeddings of H_1 and H_2 to a planar embedding of H such that the C-constraint is satisfied, it is necessary that the embedding of H_1 allows to insert an edge of color c such that the C-constraint is satisfied. Thus, it is necessary that the order of edges around v in H_1 satisfies a C^{-c}-constraint. Note that if the C^{-c}-constraint is satisfied, it is guaranteed that the embeddings of H_1 and H_2 can be merged such that the original C-constraint is satisfied. Therefore, it is necessary and sufficient to equip the copy of v in H_1 with a C^{-c}-constraint whereas the copy of v in H_2 remains unconstrained.

Note that we may assume that after a linear-time preprocessing every edge in H is labeled with the block it belongs to. Then, for a cut vertex v a split as described above takes $O(\deg(v))$-time. When no cut vertex is left, we return the resulting alternation-constrained blocks of H. ◀

Algorithm for the Embedding Problem. In the following we assume familiarity with the PQ-tree [4, 3] and SPQR-tree data structures [7]. We define a more general problem GENERAL ALTERNATION-CONSTRAINED PLANARITY (GACP) whose input is a graph H where vertices of degree 4, 5, or 6 may be equipped with an alternation constraint or with a (synchronized) PQ-tree (*but not both*). The question is whether H admits a planar embedding such that all alternation constraints are satisfied (i.e., H is feasible) and the order of edges around a vertex with a PQ-tree B is compatible with B. The 2-CONNECTED GACP problem is the restriction of GACP to input graphs that are 2-connected. Clearly, every instance of (2-CONNECTED) ACP is an instance of (2-CONNECTED) GACP. For our purpose, however, it will turn out that PQ-tree constraints are easier to handle. Thus, given an instance of ACP we aim to construct an equivalent instance of GACP, where as many alternation constraints as possible are replaced by PQ-trees. In particular, alternation constraints of degree-4 vertices can be replaced by the PQ-trees shown in Fig. 7, see the full version [5] for details. Hence we may assume from now on that no vertex with an alternation constraint in H has degree 4; i.e., all these vertices have degree 5 or 6.

Let v be a vertex with alternation constraints. We call two edges e, f incident to v a *consecutive edge pair*, if they are consecutive (around v) in *every* planar embedding of H that satisfies all constraints. In the full version [5] we show that, with the exception of K_3^{-r}, an alternation constraint at a vertex incident to a consecutive edge pair can be replaced by a PQ-tree. The overall strategy of the remaining section consists of three steps. In Step 1 we identify consecutive edge pairs in H with the help of the SPQR-tree of H and replace the corresponding alternation constraints by PQ-trees. By doing this exhaustively and using a special operation described in [5] to remove the K_3^{-r}-constraints, we end up with an instance whose alternation constraints are all K_3-constraints and every vertex with such a constraint appears in the skeletons of exactly two P-nodes and one S-node in the SPQR-tree. In Step 2, we handle such constraints by considering them on a more global scale. We show that they form cyclic structures, where either the constraints cannot be satisfied or can be dropped and satisfied irrespective of the remaining solution. Eventually, we arrive at an instance with only (synchronized) PQ-trees as constraints, which we solve with standard techniques in Step 3.

For the rest of this section let H be an instance of 2-CONNECTED GACP and let T be the SPQR-tree of H. We begin with Step 1 and identify consecutive edge pairs.

▶ **Lemma 11** (⋆). *Let H be an instance of* 2-CONNECTED GACP *and let T be the SPQR-tree of H. A vertex v with alternation constraint C in H is incident to a consecutive edge pair if*

(i) *there is a skeleton in T with a virtual edge containing exactly two edges from $E(v)$ or*

(ii) *there is a skeleton in T with a virtual edge containing all but two edges from $E(v)$ or*

(iii) *v appears in the skeleton of an R-node in T.*

Since we immediately replace alternation constraints by PQ-trees whenever we find a consecutive edge pair, we assume from now on that no vertex with alternation constraint different from K_3^{-r} satisfies one of the conditions of Lemma 11. Let μ be a node of T and let v be a vertex of its skeleton incident to the virtual edges $e_1, \dots, e_k$. Then the *distribution vector* $(d_1, \dots, d_k)$ of v, with $d_i \geq d_{i+1}$ for every $1 \leq i < k$, contains for each virtual edge e_i the number d_i of edges from $E(v)$ contained in e_i.

Consider a vertex v with alternation constraint different from K_3^{-r} in H. Assume that v appears in an S-node ν in T. Since the vertices in the skeleton of an S-node have degree 2, v also appears in at least one other node μ adjacent to ν in T. Note that μ is a P-node since there are no two adjacent S-nodes in an SPQR-tree. Hence, we may assume in the following that every vertex with alternation constraint appears in a P-node μ. Recall that the vertices in the skeleton of a P-node have degree at least 3; i.e., the edges of $E(v)$ are distributed among at least three virtual edges. Since by assumption no virtual edge contains exactly two edges from $E(v)$ or all but two edges from $E(v)$, the only possible distributions without a consecutive edge pair are $(1, 1, 1, 1, 1, 1)$, $(1, 1, 1, 1, 1)$ and $(3, 1, 1, 1)$.

In the first two cases it can be shown that we can get rid of the alternation constraint C of v as it is either always possible to reorder the children of μ according to C in a realization of H without C or H without C (and thus H) is not realizable. Similar techniques allow us to show that we can get rid of *(i)* P_3-constraints, *(ii)* K_3^{-r}-constraints and of *(iii)* K_3-constraints of poles of P-nodes such that the other pole is either unconstrained or has a PQ-tree. The proofs are deferred to the full version [5]. Hence, we may assume that only K_3-constraints occur and that every vertex v with K_3-constraint appears in the skeleton of a P-node ν in T with distribution vector $(3, 1, 1, 1)$, whose pole distinct from v also has a K_3-constraint. This concludes Step 1.

Now move to Step 2. Let v be a vertex with K_3-constraint. The three edges from $E(v)$ contained in the same virtual edge in the skeleton of ν must have pairwise distinct colors; otherwise, H is a no-instance. Since there are no two adjacent P-nodes in an SPQR-tree and by assumption no vertex with alternation constraint appears in an R-node, v also appears in an S-node with distribution vector $(3, 3)$. Let μ be an S-node in T that contains v. Since there are no two adjacent S-nodes in an SPQR-tree, for each neighbor u of v in the skeleton of μ, there is a P-node adjacent to μ in T with poles v, u. Thus, by assumption, the neighbors of v in the skeleton of μ also have a K_3-constraint. Iteratively, it follows that every vertex in the skeleton of μ has a K_3-constraint and shares a P-node with each of its two neighbors.

Consider an S-node μ in T that contains alternation-constrained vertices $v_0, \dots, v_{k-1}$ in this order; see Fig. 8. In the following, we consider the indices of the vertices and edges in μ modulo k. For every $0 \leq i < k$, we denote the virtual edge between v_i and v_{i+1} by e_i and let ν_i be the P-node adjacent to μ in T with poles v_i, v_{i+1}. Note that for every i, the virtual edge e in the skeleton of ν_i that contains three edges from $E(v_i)$ also contains three edges from $E(v_{i+1})$, since e is the virtual edge representing μ. Thus, if we fix the order of the three edges from $E(v_i)$ in e_i, this fixes the order of the three edges from $E(v_{i+1})$ in e_i.

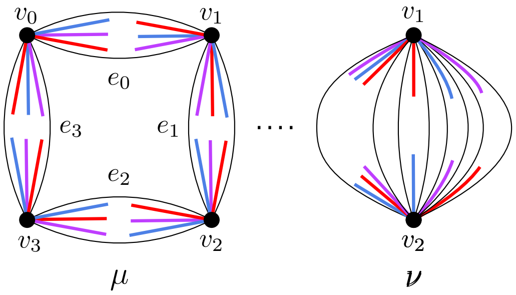


Figure 8 An S-node μ and an adjacent P-node ν.

Since v_{i+1} has an alternation constraint, this also fixes the order of the edges from $E(v_{i+1})$ in e_{i+1}. In this way, a fixed order of the three edges from $E(v_1)$ in e_1 implies an order of the edges from $E(v_{k-1})$ in e_{k-1}, which in turn implies an order on the three edges from $E(v_1)$ in e_{k-1}. If there exists an order of the three edges from $E(v_1)$ in e_1 that implies an order of the remaining edges from $E(v_1)$ in e_{k-1} such that the K_3-constraint is satisfied, we obtain an equivalent instance by removing all K_3-constraints of vertices in the skeleton of μ, since we can reorder the parallels adjacent to ν independently of the remaining graph. Otherwise, H is a no-instance. By the discussion above, we have the following.

▶ **Lemma 12.** *Let μ be an S-node in T that contains vertices with K_3-constraint. There is an $O(\deg(\mu))$-algorithm that either recognizes that H is a no-instance, or computes an equivalent instance by removing all K_3-constraints of vertices in the skeleton of μ.*

Now we may assume that our graph H does not contain alternation constraints and start with Step 3. To solve such an instance, we expand each vertex with its associated PQ-tree, if any, into a gadget that allows the same circular orders of its incident edges as the PQ-tree (essentially a P-node becomes a normal vertex, whereas a Q-node expands into a wheel as described in [13]). Embedding the resulting graph $H^\star$ and then contracting the gadgets back into single vertices already ensures that for each vertex of H the order of its incident edges is represented by its PQ-tree. Since our synchronized PQ-trees only involve Q-nodes, it then suffices to ensure that synchronized Q-nodes are flipped consistently. To this end, observe that each wheel is 3-connected and hence its embedding is determined by a single R-node in the SPQR-tree of $H^\star$. This allows us to express such constraints in a simple 2-SAT formula of linear size, similar to, e.g., [1, 13]. Therefore, we obtain the following.

▶ **Theorem 13 ($\star$).** *Let $A = (G, \mathcal{X})$ be an n-vertex AT-graph such that $\lambda(A) \leq 3$. There exists an $O(n)$-time algorithm that decides whether A is a positive instance of* SATR *and that, in the positive case, computes a simple realization of A.*

5 Conclusions and Open Problems

We proved that deciding whether an AT-graph A is simply realizable is NP-complete, already when the size $\lambda(A)$ of the largest connected components of the crossing graph $\mathcal{C}(A)$ satisfies $\lambda(A) \leq 6$. On the other hand, we described an optimal linear-time algorithm that solves the problem when $\lambda(A) \leq 3$. This is the first efficient algorithm for the Simple AT-graph Realizability problem that works on general graphs.

An open problem that naturally arises from our findings is filling the gap between tractability and intractability: What is the complexity of Simple AT-graph Realizability if $\lambda(A)$ is 4 or 5? A first issue is that Lemma 1 only allows to untangle cliques of size 3 and it is not clear whether a similar result can be proved for components of size 4. Furthermore, contracting larger crossing structures yields more complicated alternation constraints and it is not clear whether they can be turned into PQ-trees, similar to the case of components of size 3. We therefore feel that different techniques may be necessary to tackle the cases where $4 \leq \lambda(A) \leq 5$.

Another interesting direction is to study alternative structural parameters under which the problem can be tackled, and which are not ruled out by our hardness result, as discussed in the introduction; for example the vertex cover number of $\mathcal{C}(A)$. Finally, one can try to extend our approach to the "weak" setting (i.e., the WEAK AT-GRAPH REALIZABILITY problem), still requiring a simple realization.

References

1 Thomas Bläsius, Simon D. Fink, and Ignaz Rutter. Synchronized planarity with applications to constrained planarity problems. *ACM Trans. Algorithms*, 19(4):34:1–34:23, 2023. doi:10.1145/3607474.

2 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Trans. Algorithms*, 12(2):16:1–16:46, 2016. doi:10.1145/2738054.

3 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

4 Kellogg Speed Booth. *PQ-tree algorithms.* University of California, Berkeley, 1975.

5 Giordano Da Lozzo, Walter Didimo, Fabrizio Montecchiani, Miriam Münch, Maurizio Patrignani, and Ignaz Rutter. Simple realizability of abstract topological graphs. *CoRR*, abs/2409.20108, 2024. doi:10.48550/arXiv.2409.20108.

6 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

7 Giuseppe Di Battista and Roberto Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. doi:10.1007/BF01961541.

8 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.

9 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous graph embeddings with fixed edges. In Fedor V. Fomin, editor, *32nd International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2006*, volume 4271 of *Lecture Notes in Computer Science*, pages 325–335. Springer, 2006. doi:10.1007/11917496_29.

12 Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition.* Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.

13 Carsten Gutwenger, Karsten Klein, and Petra Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *J. Graph Algorithms Appl.*, 12(1):73–95, 2008. doi:10.7155/JGAA.00160.

14 Seok-Hee Hong. Beyond planar graphs: Introduction. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs, Communications of NII Shonan Meetings*, pages 1–9. Springer, 2020. doi:10.1007/978-981-15-6533-5_1.

15 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

16 Jan Kratochvíl. String graphs. II. Recognizing string graphs is NP-hard. *J. Comb. Theory, Ser. B*, 52(1):67–78, 1991. doi:10.1016/0095-8956(91)90091-W.

17 Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics*, 52(3):233–252, 1994. doi:10.1016/0166-218X(94)90143-0.

18 Jan Kratochvíl, Anna Lubiw, and Jaroslav Nesetril. Noncrossing subgraphs in topological layouts. *SIAM J. Discret. Math.*, 4(2):223–244, 1991. doi:10.1137/0404022.

19 Jan Kratochvíl and Jiří Matoušek. NP-hardness results for intersection graphs. *Commentationes Mathematicae Universitatis Carolinae*, 30(4):761–773, 1989. URL: http://eudml.org/doc/17790.

20 Jan Kratochvíl and Jirí Matousek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994. doi:10.1006/JCTB.1994.1071.

21 Jan Kyncl. Simple realizability of complete abstract topological graphs in P. *Discret. Comput. Geom.*, 45(3):383–399, 2011. doi:10.1007/S00454-010-9320-X.

22 Jan Kyncl. Simple realizability of complete abstract topological graphs simplified. *Discret. Comput. Geom.*, 64(1):1–27, 2020. doi:10.1007/S00454-020-00204-0.

23 Marcus Schaefer. Toward a theory of planarity: Hanani-tutte and planarity variants. *J. Graph Algorithms Appl.*, 17(4):367–440, 2013. doi:10.7155/JGAA.00298.

24 Marcus Schaefer, Eric Sedgwick, and Daniel Stefankovic. Recognizing string graphs in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003. doi:10.1016/S0022-0000(03)00045-X.

25 Marcus Schaefer and Daniel Stefankovic. Decidability of string graphs. *J. Comput. Syst. Sci.*, 68(2):319–334, 2004. doi:10.1016/J.JCSS.2003.07.002.

26 Ileana Streinu, Károly Bezdek, János Pach, Tamal K. Dey, Jianer Chen, Dina Kravets, Nancy M. Amato, and W. Randolph Franklin. Discrete and computational geometry. In Kenneth H. Rosen, John G. Michaels, Jonathan L. Gross, Jerrold W. Grossman, and Douglas R. Shier, editors, *Handbook of Discrete and Combinatorial Mathematics*. CRC Press, 1999. doi:10.1201/9781439832905.CH13.

Exact Algorithms for Clustered Planarity with Linear Saturators

Giordano Da Lozzo
Roma Tre University, Italy

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Siddharth Gupta
BITS Pilani, K K Birla Goa Campus, India

Bojan Mohar
Department of Mathematics, Simon Fraser University, Burnaby, Canada

Sebastian Ordyniak
University of Leeds, UK

Meirav Zehavi
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We study Clustered Planarity with Linear Saturators, which is the problem of augmenting an n-vertex planar graph whose vertices are partitioned into independent sets (called *clusters*) with paths – one for each cluster – that connect all the vertices in each cluster while maintaining planarity. We show that the problem can be solved in time $2^{\mathcal{O}(n)}$ for both the variable and fixed embedding case. Moreover, we show that it can be solved in subexponential time $2^{\mathcal{O}(\sqrt{n}\log n)}$ in the fixed embedding case if additionally the input graph is connected. The latter time complexity is tight under the Exponential-Time Hypothesis. We also show that n can be replaced with the vertex cover number of the input graph by providing a linear (resp. polynomial) kernel for the variable-embedding (resp. fixed-embedding) case; these results contrast the NP-hardness of the problem on graphs of bounded treewidth (and even on trees). Finally, we complement known lower bounds for the problem by showing that Clustered Planarity with Linear Saturators is NP-hard even when the number of clusters is at most 3, thus excluding the algorithmic use of the number of clusters as a parameter.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Computational geometry; Mathematics of computing → Graph algorithms

Keywords and phrases Clustered planarity, independent c-graphs, path saturation, graph drawing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.24

Related Version *Full Version*: `http://arxiv.org/abs/2409.19410` [39]

Funding *Giordano Da Lozzo*: Supported, in part, by MUR of Italy (PRIN Project no. 2022ME9Z78 – NextGRAAL and PRIN Project no. 2022TS4Y3N – EXPAND).
Robert Ganian: Supported by the Austrian Science Fund (FWF, Project 10.55776/Y1329) and the Vienna Science and Technology Fund (WWTF, Project 10.47379/ICT22029).
Siddharth Gupta: Supported, in part, by BITS Pilani New Faculty Seed Grant.
Bojan Mohar: Supported, in part, by the NSERC Discovery Grant R832714 (Canada), by the ERC Synergy grant (European Union, ERC, KARST, project number 101071836), and by the Research Project N1-0218 of ARIS (Slovenia).
Meirav Zehavi: Supported by the European Research Council (ERC) grant titled PARAPATH.

Acknowledgements This research started at the Dagstuhl Seminar: New Frontiers of Parameterized Complexity in Graph Drawing; seminar number: 23162; April 16-21, 2023 [28].

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 24; pp. 24:1–24:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The representation of graphs with a hierarchical structure has become an increasingly crucial tool in the analysis of networked data across various domains. Indeed, by recursively grouping vertices into clusters exhibiting semantic affinity, modern visualization tools allow for the visualization of massive graphs whose entire visualization would otherwise be impossible. Clustered graphs, where a graph's vertex set is partitioned into distinctive subsets, naturally originate in various and diverse fields such as knowledge representation [35], software visualization [41], visual statistics [10], and data mining [40], only to name a few.

Formally, a *flat clustered graph* (for short, *clustered graph* or *c-graph*) is a pair $\mathcal{C} = (G, \mathcal{V})$ where G is a graph and $\mathcal{V} = \{V_1, \dots, V_k\}$ is a partition of the vertex set of G into sets V_i called *clusters*. The graph G is the *underlying graph* of $\mathcal{C}$. A pivotal criterion for a coherent visualization of a clustered graph stems from the classical notion of graph planarity. A *c-planar drawing* of a clustered graph $\mathcal{C} = (G, \mathcal{V})$ is defined as a planar drawing of G, accompanied by a representation of each cluster $V_i \in \mathcal{V}$ as a region $\mathcal{D}_i$ homeomorphic to a closed disc, such that regions associated with different clusters are disjoint, the drawing of the subgraph of G induced by the vertices of each cluster V_i lies in the interior of $\mathcal{D}_i$, and each edge crosses the boundary of a region at most once. The problem of testing for the existence of a c-planar drawing of a clustered graph, called C-Planarity Testing, was introduced by Lengauer [37] (in an entirely different context) and then rediscovered by Feng, Cohen, and Eades [23]. Determining the complexity of the problem has occupied the agenda of the Graph Drawing community for almost three decades [2,6,7,9,11,14–16,18,19,21,26,29,30,32–34,43]. The seemingly elusive goal of settling the question regarding the computational complexity of this problem has only been recently addressed by Fulek and Tóth [27], who presented a polynomial-time algorithm running in $\mathcal{O}(n^8)$ time (and in $\mathcal{O}(n^{16})$ time for the version of the problem in which a recursive clustering of the vertices is allowed. It is worth pointing out that, even before a polynomial-time solution for the C-Planarity problem was presented, Cortese and Patrignani [17] established that the problem retains the same complexity on both flat and general (i.e., recursively clustered) instances. Subsequently, a more efficient algorithm running in quadratic time has been presented by Bläsius, Fink, and Rutter [8].

Patrignani and Cortese studied *independent c-graphs*, i.e., c-graphs where each of the clusters induces an independent set [17]. A characterization given by Di Battista and Frati [21] implies that an independent c-graph is c-planar if and only if the underlying graph can be augmented by adding extra edges, called *saturating edges*, in such a way that the resulting graph has a planar embedding and each cluster induces a tree in the resulting graph. Angelini et al. [4] considered a constrained version of c-planarity, called Clustered Planarity with Linear Saturators (for short, CPLS), which takes as input an independent c-graph and requires that each cluster induces a path (instead of a tree) in the augmented graph. They proved that the CPLS problem is NP-complete for c-graphs with an unbounded number of clusters, regardless of whether the input graph is equipped with an embedding or not. The problem fits the paradigm of augmenting planar graphs with edges in such a way that the resulting graph remains planar, while achieving some other desired property, which is a central question in Algorithmic Graph Theory [1,12,13,24,36,42].

Although CPLS is a topological problem, it stems from a geometric setting within *intersection-link representations*, a form of hybrid representations for locally-dense globally-sparse graphs [4]. Specifically, see also [3], given a c-graph $\mathcal{C} = (G, \mathcal{V})$ whose every cluster induces a clique, the Clique Planarity problem asks to compute a *clique planar representation* of $\mathcal{C}$, i.e., a drawing of $\mathcal{C}$ in which each vertex $v \in V(G)$ is represented as a

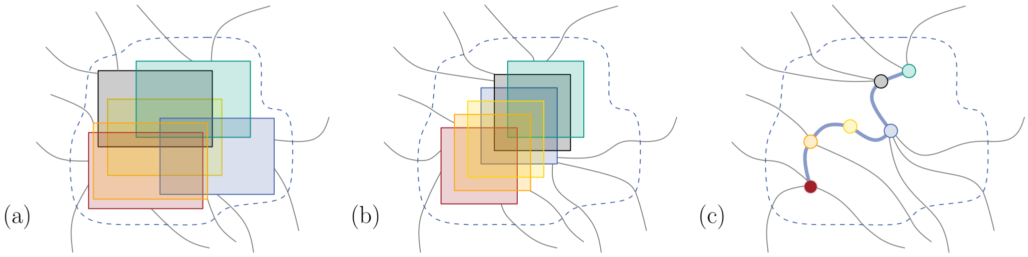


Figure 1 (a) A partial clique planar representation focused on a cluster. (b) Canonical representation of the cluster in (a). A linear saturation of the vertices of the cluster corresponding to (b).

translate $R(v)$ of a given rectangle R, each intra-cluster edge (u, v) is represented by an intersection between $R(u)$ and $R(v)$, and each inter-cluster edge (u, v) is represented by a Jordan arc connecting the boundaries of $R(u)$ and $R(v)$ that intersects neither the interior of any rectangle nor the representation of any other inter-cluster edge. Notably, the authors showed that a c-graph whose every cluster induces a clique admits a clique planar representation if and only if it admits a so-called *canonical representation*, where the vertices are squares arranged in a "linear fashion"; see Fig. 1. This allowed them to establish the equivalence between CPLS and Clique Planarity. In particular, they proved that a c-graph $\mathcal{C}$ whose every cluster induces a clique is a yes-instance of Clique Planarity if and only if the c-graph obtained by removing all intra-cluster edges from $\mathcal{C}$ is a yes-instance of CPLS.

Our Contribution. In this paper, we study the CPLS problem from a computational perspective, in both the fixed embedding as well as the variable embedding setting. In the fixed embedding case, the underlying graph of the c-graph comes with a prescribed combinatorial embedding, which must be preserved by the output drawing. Instead, in the variable embedding setting, we are allowed to select the embedding of the underlying graph. To distinguish these two settings, we refer to the former problem (i.e., where a fixed embedding is provided as part of the input) as CPLSF. Our main results are as follows.
(1) In Section 3 we give exact single-exponential and sub-exponential algorithms for the problems. In particular, both CPLS and CPLSF can be solved in $2^{\mathcal{O}(n)}$ time. Moreover, we obtain a subexponential $2^{\mathcal{O}(\sqrt{n}\log n)}$ algorithm for CPLSF when the underlying graph is connected; this result is essentially tight under the Exponential Time Hypothesis [31]. In both cases, the main idea behind the algorithms is to use a divide-and-conquer approach that separates the instance according to a hypothetical separator in the solution graph.
(2) In Section 4 we obtain polynomial kernels (and thus establish fixed-parameter tractability) for both CPLS and CPLSF with respect to the vertex cover number of the underlying graph. Interestingly, while being provided with an embedding allowed us to design a more efficient exact algorithm for CPLSF, in the parameterized setting the situation is reversed: we obtain a linear kernel for CPLS, but for CPLSF the size of the kernel is cubic in the vertex cover number. Combining the former result with our exact algorithm for CPLS allows us to obtain an algorithm that runs in single-exponential time with respect to the vertex cover number.
(3) In Section 5 we observe that the CPLS problem is NP-complete on trees and even a disjoint union of stars. Since stars have treedepth, pathwidth, and treewidth one, this charts an intractability border between the vertex cover number parameterization used in Section 4 and other parameters. Then we prove that the problem is NP-complete even for c-graphs having at most 3 clusters, thus strengthening the previously known hardness result for an

unbounded number of clusters. This result combined with the equivalence between CPLS and Clique Planarity shows that Clique Planarity is NP-complete for instances with a bounded number of clusters, which solves an open problem posed in [3, OP 4.3].

Full proofs and further details for paragraphs marked with ($\star$) can be found in the full version of the paper [39].

2 Preliminaries

For a positive integer k, we denote by $[k]$ the set $\{1, \ldots, k\}$. We use standard terminology in the context of graph theory [22] and graph drawing [20]. An *embedded graph* $G_\mathcal{E}$ is a planar graph G equipped with an embedding $\mathcal{E}$. A *noose* N of an embedded graph $G_\mathcal{E}$ is a simple closed curve in some drawing Γ of $G_\mathcal{E}$ that

(i) intersects G only at vertices and

(ii) traverses each face of Γ at most once.

Given a subgraph H of G, we denote by $\mathcal{E}(H)$ the embedding of H obtained from $\mathcal{E}$ by restricting it to H. The *vertex cover number* of a graph G is the smallest size of a vertex cover in G. We assume basic familiarity with the parameterized complexity framework, and in particular with the notion of *kernelization* [25].

Clustered Planarity with Linear Saturators. Let $\mathcal{C}$ be a c-graph. We say that $\mathcal{C}$ has a *fixed embedding* if the underlying graph of $\mathcal{C}$ is an embedded graph, and has a *variable embedding* otherwise. We say that $\mathcal{C}$ is an *embedded c-graph* if $\mathcal{C}$ has a fixed embedding.

Let $\mathcal{C} = (G, \{V_1, \ldots, V_k\})$ be an independent c-graph, i.e., for every $i \in [k]$, V_i is an independent set of G. We say that G can be *linearly saturated* if there exist sets $Z_1, \ldots, Z_k$ of non-edges of G such that

(i) $H = (V(G), E(G) \cup Z)$ for $Z = \bigcup_{i=1}^{k} Z_i$ is planar,

(ii) for every $i \in [k]$, each edge in Z_i connects two vertices of V_i in H, and

(iii) for every $i \in [k]$, the graph $H[V_i]$ is a path.

For $i \in [k]$, the edges in Z_i are the *saturating edges* of cluster V_i, and H is the *linear saturation* of G *via* $Z_1, \ldots, Z_k$. We now define the Clustered Planarity with Linear Saturators (for short, CPLS) and the Fixed Embedding Clustered Planarity with Linear Saturators (for short, CPLSF) problems.

▶ **CPLS:** Given an independent c-graph $(G, \mathcal{V})$, does there exist a linear saturation of G?

▶ **CPLSF:** Given an independent embedded c-graph $(G_\mathcal{E}, \mathcal{V})$, does there exist a linear saturation H of G that admits an embedding $\mathcal{E}'$ for which $\mathcal{E}'(G)$ coincides with $\mathcal{E}$?

To devise exact algorithms for the CPLS and CPLSF problem, it will be useful to consider a more general setting. A c-graph is *paths-independent* if each of its clusters induces a collection of paths; the notion of a linear saturator for a paths-independent c-graph is the same as that of an independent c-graph. We now define the Clustered Planarity with Linear Saturators Completion (for short, CPLS-Completion) and the Fixed Embedding Clustered Planarity with Linear Saturators Completion (for short, CPLSF-Completion) problem as the generalizations of CPLS and CPLSF, respectively, where the input is a paths-independent c-graph. Observe that in case of CPLS (resp., CPLSF), $H[V_i] = H[Z_i]$ as every cluster induces an independent set in G; this is, however, not necessarily true in the case of CPLS-Completion (resp., CPLSF-Completion).

3 Exact Algorithms for CPLS and CPLSF

This section details our exact single- and sub-exponential algorithms for CPLS and CPLSF.

Proof Ideas. We aim to solve the problem via a divide-and-conquer approach, where at each iteration, we "split" the current instance of the problem into simpler (and, in particular, substantially smaller) sub-instances of the problem. To understand how to perform the split, consider an (unknown) solution Z, and the graph $H = (V(G), E(G) \cup Z)$. As this graph is planar, there exists a noose N that intersects only $\mathcal{O}(\sqrt{|V(G)|})$ vertices of H and does not intersect any edges of H, such that both the interior and the exterior of N contain a constant fraction (roughly between 1/3 to 2/3) of the vertices of G [38]. Thus, naturally, we would like to split our problem instance into two instances, one corresponding to the interior and boundary of N, and the other corresponding to the exterior and boundary of N (so, the boundary is common to both). However, two issues arise, which we describe next.

The first (simpler) issue is that we do not know N since we do not know Z. However, since N intersects only few vertices, we can simply "guess" N by guessing the set U of intersected vertices, and the cyclic order ρ in which N intersects them. By guessing, we mean that we iterate over all possible options, and aim to find at least one that yields a solution (if a solution exists). Having U and ρ at hand, we still do not know the interior and exterior of N, and therefore, we still do not know how to create the two simpler sub-instances. Thus, we perform additional guesses: We guess the set I of vertices drawn (with respect to the planar drawing of H) strictly inside N and thereby also the set O of vertices drawn strictly outside N. Additionally, for the set of edges having both endpoints in U, we guess a partition $\{E_{\mathsf{in}}, E_{\mathsf{out}}\}$ that encodes which of them are drawn inside N and which of them are drawn outside N. Specifically for the fixed embedding case, we non-trivially exploit the given embedding to perform the guesses of I, O and $\{E_{\mathsf{in}}, E_{\mathsf{out}}\}$ in a more sophisticated manner that yields only subexponentially many guesses.

The second (more complicated) issue is that we cannot just create two instances: One for the subgraph of G induced by $I \cup U$ (and without the edges in E_{out}) and the other for the subgraph of G induced by $O \cup U$ (and without the edges in E_{in}) and solve them independently. The two main concerns are the following: First, we need a single planar drawing for the entire (unknown) graph H and so the drawings of the two graphs in the two sub-instances should be "compatible". Second, we may not need to (and in some cases, in fact, must not) add edges to a graph in any of the two sub-instances to connect all vertices in the same cluster in that graph into a single path. Instead, we need to create a collection of paths, so that the two collections that we get, one for each of the two sub-instances, will together yield a single path.

To handle the second concern, we perform additional guesses. Specifically, we guess some information on how the (solution) cluster paths in H "behave" when they are restricted to the interior of N – we guess a triple (M, P, D) which encodes, roughly speaking, a pairing M between some vertices in U that are connected by the cluster paths only using the interior of N, the set of vertices P through which the cluster paths enter the interior of N and "do not return", and the set of vertices D that are incident to two edges in Z drawn in the interior. Now, having such a guess at hand, we handle both concerns by defining a special graph that augments the graph induced by $G[U \cup O]$ (with the edges in E_{in} removed) so that the solutions returned for the corresponding sub-instance will have to be, in some sense, "compatible" with (M, P, D) as well as draw O and E_{out} outside N while preserving the order ρ. The definition of this augmented graph is, perhaps, the most technical definition required for the proof, as it needs to handle both concerns simultaneously. Among other operations performed to obtain this augmented graph, for the first concern, we add extra edges between vertices in M, attach pendants on P, and treat vertices in D as if they belong to their own clusters, and for the second concern, we triangulate the result in a careful manner.

The Variable-Embedding Case. Towards solving CPLS, we recursively solve the more general problem mentioned earlier, namely, CPLS-Completion. Additionally, we suppose that some of the vertices of the input graph can be marked, and that we are not allowed to add edges incident to marked vertices. To avoid confusion, we will denote this annotated version by CPLS-COMPLETION*. The rest of this section is devoted to the proof of the following.

▶ **Theorem 1.** *Let $\mathcal{C} = (G, \mathcal{V})$ be an n-vertex paths-independent c-graph. It can be tested whether $\mathcal{C}$ is a positive instance of* CPLS-COMPLETION* *in $8^{n+\mathcal{O}(\sqrt{n}\log n)} = 2^{\mathcal{O}(n)}$ time.*

We start with the following definition.

▶ **Definition 2 (Non-Crossing Matchings and the Partition MatPenDel).** *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. Let ρ be a cyclic ordering of some subset U of $V(G)$. A matching M is a* non-crossing matching *of ρ if it is a matching in the graph $H = (U, \{\{a, b\} : a, b \in U\})$ such that for every pair of edges $\{a, b\}, \{c, d\} \in M$, when we traverse U in the cyclic order ρ, starting with a, we either encounter b before both c and d, or we encounter both c and d before b. Denote by* $\mathrm{MatPenDel}(\mathcal{V}, \rho)$ *(which stands for* Matching, Pendants and Deleted*) the set of all triples (M, P, D) such that:*

- *M is a non-crossing matching of ρ such that each edge of M matches only vertices in the same cluster, and*
- *$P, D \subseteq U \setminus V(M)$ are disjoint sets, where $V(M)$ is the set of vertices matched by M.*

Intuitively, ρ will represent a cyclic balanced separator of G, i.e., a noose in a drawing of the solution graph that separates the solution graph into two almost equally sized subgraphs, M will represent path segments between pairs of vertices on ρ, P ("pendants") will represent vertices through which the paths leave ρ never to return,[1] and D will represent degree-2 vertices on the aforementioned path segments (which will be, in a sense, deleted when we "complement" the triple). This will be formalized in Definition 5 ahead.

▶ **Observation 3.** *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. Let ρ be a cyclic ordering of some subset U of $V(G)$. Then, $|\mathrm{MatPenDel}(\mathcal{V}, \rho)| = 2^{\mathcal{O}(|U|)}$. Moreover, the set $\mathrm{MatPenDel}(\mathcal{V}, \rho)$ can be computed in time $2^{\mathcal{O}(|U|)}$.*

We formalize the notion of a *partial solution* as follows.

▶ **Definition 4 (Partial Solution).** *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. A* cluster path *is a path all of whose vertices belong to the same cluster in $\mathcal{V}$. A* partial solution *for $\mathcal{C}$ is a set $\mathcal{S}$ of vertex-disjoint cluster paths. Note that $\mathcal{S}$ can contain several cluster paths whose vertices belong to the same cluster. We say that $\mathcal{S}$ is* properly marked *if every edge in $\mathcal{S}$ incident to a marked vertex also belongs to G.*

Next, we formalize how a triple in MatPenDel captures information on a partial solution (see Fig. 2a). For intuition, think of $\mathcal{S}$ as if it consisted of paths that contain vertices only from the exterior (or only from the interior) of a cyclic separator.

[1] Thus, for a (non-partial) solution, P contains at most 2 vertices per cluster, though we do not need to formally demand this already in Definition 5.

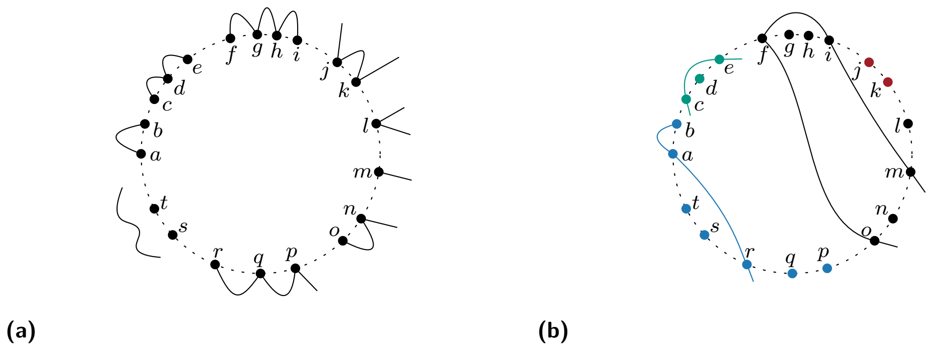


Figure 2 (a) Example for Definition 5: The paths in $\mathcal{S}$ are drawn as black curves, and the vertices in U are marked by disks. We have $M = \{\{a,b\},\{c,e\},\{f,i\}\}, P = \{m,o,r\}, D = \{d,g,h,j,k,l,n,p,q\}$ and $U \setminus (V(M) \cup P \cup D) = \{s,t\}$. (b) Example for Definition 7: The vertices in U are marked by disks, and their association with the clusters is indicated by colors. Suppose $M_{\mathsf{in}} = \{\{a,b\},\{c,e\},\{f,i\}\}, P_{\mathsf{in}} = \{m,o,r\}, D_{\mathsf{in}} = \{d,g,h,j,k,l,n,p,q\}$ and $U \setminus (V(M_{\mathsf{in}}) \cup P_{\mathsf{in}} \cup D_{\mathsf{in}}) = \{s,t\}$, and $M_{\mathsf{out}} = \{\{f,o\},\{i,m\},\{r,a\}\}$, $P_{\mathsf{out}} = \{c,e\}$, $D_{\mathsf{out}} = \{s,t\}$ and $U \setminus (V(M_{\mathsf{out}}) \cup P_{\mathsf{out}} \cup D_{\mathsf{out}}) = \{b,d,g,h,j,k,l,n,p,q\}$. Then, the triples $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}})$ and $T_{\mathsf{out}} = (M_{\mathsf{out}}, P_{\mathsf{out}}, D_{\mathsf{out}})$ are complementary, and $G_{T_{\mathsf{in}},T_{\mathsf{out}}}$ is a subgraph of the illustrated graph induced by $\{a,b,c,e,f,i,m,o,r\}$. The pendants are the endpoints of the edges going inside or outside the cycle (the vertex b, for example, is not adjacent to a pendant, while the vertex c is).

▶ **Definition 5** (**Extracting a Triple from a Partial Solution**). *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. Let $U \subseteq V(G)$ and let $\mathcal{S}$ be a partial solution. Then,* $\mathrm{ExtractTriple}(U, \mathcal{S}) = (M, P, D)$ *is defined as follows:*

- *M has an edge between the endpoints of every path in $\mathcal{S}$ that has both endpoints in U,*
- *$P \subseteq U$ consists of the vertices of degree 1 in $\mathcal{S}$ belonging to U that are not in $V(M)$,*[2] *and*
- *$D \subseteq U$ consists of the vertices of degree 2 in $\mathcal{S}$ belonging to U.*

Further, $\mathcal{S}$ is compatible with *a cyclic ordering ρ of U if M is a non-crossing matching of ρ.*

We have the following immediate observation, connecting Definitions 2 and 5.

▶ **Observation 6.** *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. Let ρ be a cyclic ordering of some subset U of $V(G)$. Let $\mathcal{S}$ be a partial solution compatible with ρ. Then,* $\mathrm{ExtractTriple}(U, \mathcal{S}) \in \mathrm{MatPenDel}(\mathcal{V}, \rho)$.

We will be interested, in particular, in triples which are complementary (see Fig. 2b), which intuitively means that partial solutions for the inside and the outside described by the two triples can be combined to a solution for the whole graph:

▶ **Definition 7** (**Complementary Triples in MatPenDel**). *Let $\mathcal{C} = (G, \mathcal{V})$ be a paths-independent c-graph. Let ρ be a cyclic ordering of some subset U of $V(G)$. Then, $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}}), T_{\mathsf{out}} = (M_{\mathsf{out}}, P_{\mathsf{out}}, D_{\mathsf{out}}) \in \mathrm{MatPenDel}(\mathcal{V}, \rho)$ are* complementary *if:*

1. *$D_{\mathsf{in}} \subseteq U \setminus (V(M_{\mathsf{out}}) \cup P_{\mathsf{out}} \cup D_{\mathsf{out}})$, and $D_{\mathsf{out}} \subseteq U \setminus (V(M_{\mathsf{in}}) \cup P_{\mathsf{in}} \cup D_{\mathsf{in}})$.*[3]
2. *Let $G_{T_{\mathsf{in}},T_{\mathsf{out}}}$ denote the graph on vertex set $V(M_{\mathsf{out}}) \cup P_{\mathsf{out}} \cup V(M_{\mathsf{in}}) \cup P_{\mathsf{in}}$ and edge set $M_{\mathsf{in}} \cup M_{\mathsf{out}}$, such that for every vertex in P_{in}, and similarly, for every vertex in P_{out}, we*

[2] In other words, for every path in $\mathcal{S}$ having precisely one endpoint in U, P contains the endpoint in U.

[3] The reason why we write $\subseteq$ rather than $=$ is that some vertices in, e.g., $U \setminus (V(M_{\mathsf{out}}) \cup P_{\mathsf{out}} \cup D_{\mathsf{out}})$ can be the endpoints of solution paths. For example, consider the vertex b in Fig. 2b.

add a new vertex attached to it and belonging to the same cluster.[4] *Then, this graph is a collection of paths, such that all vertices in its vertex set that belong to the same cluster in $\mathcal{V}$ also belong to the same (single) path, and all vertices that belong to the same path also belong to the same cluster.*

The utility of complementary triples is in the following definition, which specifies when a partial solution $\mathcal{S}$ for the inner part satisfying $\mathrm{ExtractTriple}(U, \mathcal{S}) = T_{\mathsf{out}}$ can be combined with any partial solution $\mathcal{S}'$ for the outer part that satisfies $\mathrm{ExtractTriple}(U, \mathcal{S}') = T_{\mathsf{in}}$.

▶ **Definition 8 (Partial Solution Compatible with $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$).** *Consider $(G, \mathcal{V})$, a cyclic ordering ρ of some $U \subseteq V(G)$, $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}}) \in \mathrm{MatPenDel}(\mathcal{V}, \rho)$, $I \subseteq V(G) \setminus U$, and $E_{\mathsf{in}} \subseteq E(G[U])$. Then, a partial solution $\mathcal{S}$ is* compatible *with $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$ if:*

1. $\mathrm{ExtractTriple}(U, \mathcal{S}) = T_{\mathsf{out}}$ *and T_{in} are complementary.*
2. *Let $G'_{\mathsf{in}} = G[I \cup U] - (E(G[U]) \setminus E_{\mathsf{in}})$ and $G_{\mathsf{in}} = G'{}_{\mathsf{in}} - D_{\mathsf{in}}$. We have that $\mathcal{S}$ contains all and only the vertices in G_{in}, and all (but not necessarily only)*[5] *edges in G_{in} between vertices in the same cluster.*
3. *There exists a planar drawing φ_{in} of $G'_{\mathsf{in}} \cup E(\mathcal{S})$ with an inner-face whose boundary contains U (with, possibly, other vertices) ordered as by ρ.*
4. *Each path in $\mathcal{S}$ satisfies one of the following conditions: (a) it consists of all vertices of a cluster in $\mathcal{V}$ that belong to G_{in}, and has no endpoint in U, or (b) it has an endpoint in U.*

Towards the statement that will show the utility of compatibility (in Lemma 10 ahead), we need one more definition, which intuitively provides necessary conditions for obtaining a solution for $(G, \mathcal{V})$ from a partial solution that is compatible with $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$.

▶ **Definition 9 (Sensibility of $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$).** *Consider $(G, \mathcal{V})$, a cyclic ordering ρ of some $U \subseteq V(G)$, $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}}) \in \mathrm{MatPenDel}(\mathcal{V}, \rho)$, $I \subseteq V(G) \setminus U$, and $E_{\mathsf{in}} \subseteq E(G[U])$. Then, $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$ is* sensible *if:*

1. *There is no edge $\{u, v\} \in E(G)$ with $v \in I$ and $u \in O$ for $O = V(G) \setminus (I \cup U)$.*
2. *No vertex in D_{in} is adjacent in $G'_{\mathsf{in}} = G[I \cup U] - (E(G[U]) \setminus E_{\mathsf{in}})$ to a vertex in the same cluster in $\mathcal{V}$. Additionally, no vertex in $U \setminus (V(M_{\mathsf{in}}) \cup P_{\mathsf{in}} \cup D_{\mathsf{in}})$ is adjacent in $G'_{\mathsf{out}} = G[O \cup U] - E_{\mathsf{in}}$ for $O = V(G) \setminus (I \cup U)$ to a vertex in the same cluster in $\mathcal{V}$.*
3. *No cluster in $\mathcal{V}$ has non-empty intersection with both $I \cup (U \setminus D_{\mathsf{in}})$ and $O \cup D_{\mathsf{in}}$ but not with $V(M_{\mathsf{in}}) \cup P_{\mathsf{in}}$.*

We show that compatible solutions yield solutions to CPLS-Completion* in MatPenDel.

▶ **Lemma 10.** *Consider $(G, \mathcal{V})$, a cyclic ordering ρ of some $U \subseteq V(G)$, $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}}) \in \mathrm{MatPenDel}(\mathcal{V}, \rho)$, $I \subseteq V(G) \setminus U$, and $E_{\mathsf{in}} \subseteq E(G[U])$. Suppose that $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$ is sensible. Additionally, consider*

- *a properly marked partial solution $\mathcal{S}_{\mathsf{in}}$ compatible with $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$, and*
- *a properly marked partial solution $\mathcal{S}_{\mathsf{out}}$ compatible with* $\mathrm{ExtractTriple}(U, \mathcal{S}_{\mathsf{in}}) = (T_{\mathsf{out}}, O, E_{\mathsf{out}})$ *where $O = V(G) \setminus (I \cup U)$ and $E_{\mathsf{out}} = E(G[U]) \setminus E_{\mathsf{in}}$.*

Then, $Z = E(\mathcal{S}_{\mathsf{in}} \cup \mathcal{S}_{\mathsf{out}}) \setminus E(G)$ is a solution to $(G, \mathcal{V})$ as an instance of CPLS-Completion*.

The crucial tool used to partition an instance of CPLS-Completion* into two smaller instances is the *augmented graph* (AugmentGraph$((G, \mathcal{V}), \rho, (M, P, D))$), which is illustrated in Fig. 3 and intuitively ensures that a solution for the given graph is compatible with the triple (T, P, D). The central statement about augmented graphs is the following.

[4] So, if a vertex belongs to both P_{in} and P_{out}, we attach two new vertices to it.

[5] As the paths in $\mathcal{S}$ can contain edges that are not edges in G.

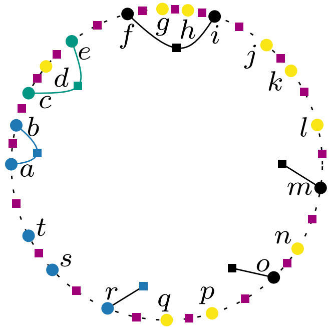


Figure 3 Example of an augmented graph. The vertices in U are marked by disks, and the clusters in $\mathcal{V}$ are $\{\{a,b,p,q,r,s,t\},\{c,d,e\},\{f,g,h,i,l,m,n,o\},\{j,k\}\}$. Suppose $M = \{\{a,b\},\{c,e\},\{f,i\}\}, P = \{m,o,r\}, D = \{d,g,h,j,k,l,n,p,q\}$ and $U \setminus (V(M) \cup P \cup D) = \{s,t\}$. Newly added vertices to the augmented graph are marked by squares, vertices belonging to their own singleton clusters are yellow or purple, and the other clusters are: (i) a,b,r,s,t and the two neighboring squares drawn inside the cycle (blue); (ii) $\{c,e\}$ and the neighboring square drawn inside the cycle (green); (iii) $\{f,i,m,o\}$ and the three neighboring squares drawn inside the cycle (black).

▶ **Lemma 11.** *Consider $(G,\mathcal{V})$, a cyclic ordering ρ of some $U \subseteq V(G)$, $T_{\mathsf{in}} = (M_{\mathsf{in}}, P_{\mathsf{in}}, D_{\mathsf{in}}) \in$ $\mathrm{MatPenDel}(\mathcal{V},\rho)$, $I \subseteq V(G) \setminus U$, and $E_{\mathsf{in}} \subseteq E(G[U])$. Let $Z_{\mathsf{in}} \neq$NULL[6] be a solution to the instance $(G_A, \mathcal{V}_A) =$*AugmentGraph*$(G'_{\mathsf{in}}, \rho, T_{\mathsf{in}})$ of* CPLS-Completion* *(where $G'_{\mathsf{in}} = G[I \cup U] - (E(G[U]) \setminus E_{\mathsf{in}})$). Then, in polynomial time, we can compute a partial solution compatible with $(T_{\mathsf{in}}, I, E_{\mathsf{in}})$ that is marked properly.*

Using the ideas outlined at the beginning of the section, we can now use Lemmas 10 and 11 to show Theorem 1.

Fixed Embedding. By building on the ideas for the variable-embedding case, we also provide a single-exponential algorithm for CPLSF, which becomes subexponential if additionally the input graph is connected. While the single-exponential algorithm is almost identical to our algorithm for CPLS, the subexponential algorithm uses connectivity together with the fixed embedding to reduce the number of guesses during the initial phase of the algorithm.

▶ **Theorem 12.** CPLSF-Completion *(and thus also* CPLSF*) can be solved in time $2^{\mathcal{O}(n)}$. Moreover, it can be solved in time $2^{\mathcal{O}(\sqrt{n}\log n)}$ if the input graph is connected.*

4 The Kernels

In this section we provide kernelization algorithms for CPLS and CPLSF parameterized by the vertex cover number k of the input graph G. Assume that X is a vertex cover of G of size k; we will deal with computing a suitable vertex cover in the proofs of the main theorems of this section. As our first step, we construct the set Z consisting of the union of X with all vertices of degree at least 3 in G. Since G can be assumed to be planar, we have:

▶ **Lemma 13** ([25, Lemma 13.3]). *$|Z| \leq 3k$.*

[6] We use NULL to algorithmically represent the non-existence of a solution (particularly in pseudocode).

Note that each vertex in $V(G) \setminus Z$ now has 0, 1 or 2 neighbors in Z. For each subset $Q \subseteq Z$ of size at most 2, let the *neighborhood type* T_Q consist of all vertices in $V(G) \setminus Z$ whose neighborhood in Z is precisely Q. Moreover, for $i \in \{0, 1, 2\}$ we let $T_i = \bigcup_{Q \subseteq Z, |Q|=i} T_Q$ contain all vertices outside of Z with degree i. At this point, our approach for dealing with CPLS and CPLSF will diverge.

The Fixed-Embedding Case. We will begin by obtaining a handle on vertices with precisely two neighbors in Z. However, to do so we first need some specialized terminology. To make our arguments easier to present, we assume w.l.o.g. that the input instance $\mathcal{I}$ is equipped with a drawing D of G that respects the given embedding.

Let G_2 be the subgraph of G induced on $Z \cup T_2$, where T_2 is the set of all vertices in $V(G) \setminus Z$ with precisely two neighbors in Z. Let D_2 be the restriction of D to G_2, and observe that D_2 only differs from D by omitting some pendant and isolated vertices. Let a face in D_2 be *special* if it is incident to more than 2 vertices of Z, and *clean* otherwise; notice that the boundary of a clean face must be a C_4 which has precisely two vertices of Z that lie on opposite sides of the C_4 and which contains only vertices in $T_0 \cup T_1$ in its interior.

▶ **Lemma 14.** *The number of special faces in D_2 is upper-bounded by $9k$, and the number of vertices in T_2 that are incident to at least one special face is upper-bounded by $36k$.*

For a pair $\{a, b\} \subseteq Z$, we say that a set P of clean faces is an *(ab)-brick* if (1) the only vertices of Z they are incident to are a and b, and (2) P forms a connected region in D_2, and (3) P is maximal with the above properties. Since the boundaries of every clean face in P consists of a, b, and two degree-2 vertices, this in particular implies that the clean faces in P form a sequence where each pair of consecutive faces shares a single degree-2 vertex; this sequence may either be cyclical (in the case of a and b not being incident to any special faces; we call such bricks *degenerate*), or the first and last clean face in P are adjacent to special faces. Observe that a degenerate brick may only occur if $|Z| = 2$.

▶ **Observation 15.** *The total number of bricks in D_2 is upper-bounded by $24k$.*

In the next lemma, we use Observation 15 to guarantee the existence of a brick with sufficiently many clean faces to support a safe reduction rule.

▶ **Lemma 16.** *Assume $|T_2| \geq 420k + 1$, where k is the size of a provided vertex cover of G. Then we can, in polynomial time, either correctly determine that $\mathcal{I}$ is a no-instance or find a vertex $v \in T_2$ with the following property: $\mathcal{I}$ is a yes-instance if and only if so is the instance $\mathcal{I}'$ obtained from $\mathcal{I}$ by removing v.*

Our next goal will be to reduce the size of T_1. To do so, we will first reduce the total number of clusters occurring in the instance – in particular, while by now we have the tools to reduce the size of G_2 (and hence also the number of clusters intersecting $V(G_2) = T_2 \cup Z$), there may be many other clusters that contain only vertices in T_1 and T_0. Let V_i be a cluster which does not intersect $V(G_2) = T_2 \cup Z$. Observe that if V_i contains vertices in more than a single face of D_2, then $\mathcal{I}$ must be a no-instance; for the following, we shall hence assume that this is not the case. In particular, for a cluster V_i such that all of its vertices are contained in a face f of D, we define its *type* $t(i)$ as follows. t_i is the set which contains f as well as all vertices v on the boundary of f fulfilling the following condition: each $v \in t(i)$ is adjacent to a pendant vertex $a \in V_i$ where a is drawn in f. An illustration of types is provided in Fig. 4.

For the next lemma, let τ be the set of all types occurring in $\mathcal{I}$.

▶ **Lemma 17.** *If $|V(G_2)| = \alpha$ and D_2 has β faces, then $|\tau| \leq 7\alpha + 8\beta$ or $\mathcal{I}$ is a no-instance.*

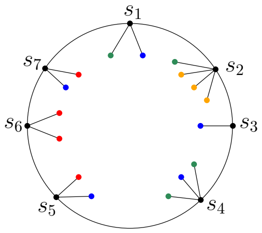


Figure 4 An illustration of cluster types in a depicted face f. In this example, individual clusters are marked by colors and none of the vertices $s_1, \ldots, s_7$ belong to any of the colored clusters. The types of the red, blue, yellow and green clusters are $\{f, s_5, s_6, s_7\}$, $\{f, s_1, s_3, s_4, s_5, s_7\}$, $\{f, s_2\}$ and $\{f, s_1, s_2, s_4\}$, respectively. Note that the depicted example cannot occur in a yes-instance since the curves for, e.g., the blue and red clusters would need to cross each other.

Lemma 17 allows us to bound the total number of cluster types in the instance via Lemma 18 below. The proof relies on a careful case analysis that depends on the size of the cluster types of the considered clusters; for cluster types of size at least 4 we directly obtain a contradiction with planarity, but for cluster types of size 2 or 3 we identify "rainbow patterns" that must be present and allow us to simplify the instance.

▶ **Lemma 18.** *Assume there are three distinct clusters, say V_1, V_2 and V_3, which do not intersect $V(G_2)$ and all have the same type of size at least 2. Then we can, in polynomial time, either correctly identify that $\mathcal{I}$ is a no-instance or find a non-empty set $A \subseteq T_1$ with the following property: $\mathcal{I}$ is a yes-instance iff so is the instance $\mathcal{I}'$ obtained from $\mathcal{I}$ by removing A.*

Having bounded the number of cluster types (Lemma 17) and the number of clusters of each type (Lemma 18), it remains to bound the number of vertices in each of the clusters.

▶ **Lemma 19.** *Let $a \in V(G_2)$ and f be a face of D_2 incident to a, and let k be the size of a provided vertex cover of G. Assume a cluster V_i contains at least $2k+3$ vertices in T_1 that are adjacent to a and lie in f. Then we can, in polynomial time, either correctly identify that $\mathcal{I}$ is a no-instance, or find a vertex $q \in V_i \cap T_1$ such that $\mathcal{I}$ is a yes-instance if and only if so is the instance $\mathcal{I}'$ obtained by deleting q.*

We now have all the ingredients required to obtain our polynomial kernel:

▶ **Theorem 20.** CPLSF *has a cubic kernel when parameterized by the vertex cover number.*

The Variable-Embedding Case. For CPLS, we establish the following result:

▶ **Theorem 21** ($\star$)**.** CPLS *has a linear kernel when parameterized by the vertex cover number.*

Proof Sketch. One can immediately observe that the vertices in T_0 are irrelevant. On a high level, we can then proceed by devising reduction rules that result in a new instance that does not contain large clusters; however it may still happen that there are many clusters occurring in the instance. To deal with this, we group all the clusters together depending on the "neighborhood types" of the vertices they contain. A second level of analysis and reduction then allows us to deal with each group of clusters; there, the most difficult case surprisingly arises when dealing with instances containing many clusters, each consisting of precisely two pendant vertices. ◀

As an immediate consequence of Theorem 21 and Theorem 1, we can obtain an improved asymptotic running time for solving CPLS:

▶ **Corollary 22.** CPLS *can be solved in time* $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, *where k and n are the vertex cover number and the number of vertices of the input graph, respectively.*

5 NP-completeness

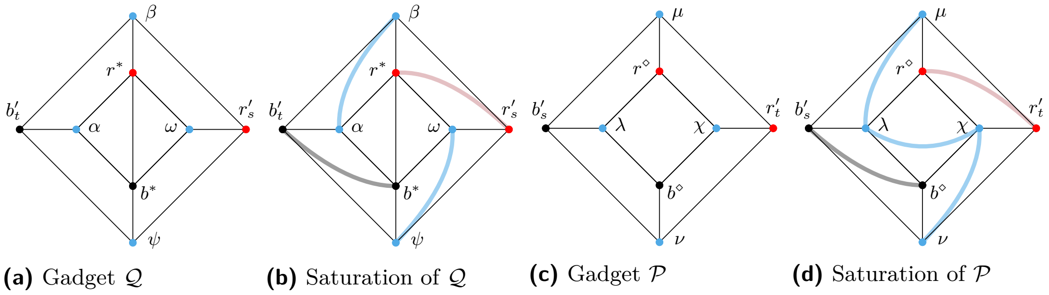
The CPLS problem was shown NP-complete for instances with an *unbounded* number of clusters, even when the underlying graph is a subdivision of a 3-connected planar graph [4]. It is a simple exercise to see that CPLS is NP-complete for trees and for forests of stars (see the Full Version). We establish the NP-completeness of CPLS and CPLSF when restricted to instances with up to three clusters, which we refer to as CPLS-3 and CPLSF-3, respectively.

The proof is based on a reduction from a specialized and still NP-complete version of the Bipartite 2-Page Book Embedding (B2PBE) problem [5]. Let $G = (U_b, U_r, E)$ be a bipartite planar graph, where the vertices in U_b and U_r are called *black* and *red*, respectively. A *bipartite* 2*-page book embedding* of G is a planar embedding $\mathcal{E}$ of G in which the vertices are placed along a Jordan curve $\ell_{\mathcal{E}}$, called *spine* of $\mathcal{E}$, the black vertices appear consecutively along $\ell_{\mathcal{E}}$, and each edge lies entirely in one of the two regions of the plane, called *pages*, bounded by $\ell_{\mathcal{E}}$. The B2PBE problem asks whether a given bipartite graph admits a bipartite 2-page book embedding. We will reduce from the B2PBE with Prescribed End-vertices (B2PBE-PE) problem. Given a bipartite planar graph $G = (U_b, U_r, E)$ and a quadruple $(b_s, b_t, r_s, r_t) \in U_b \times U_b \times U_r \times U_r$, B2PBE-PE asks whether G admits a bipartite 2-page book embedding $\mathcal{E}$ in which the vertices b_s, b_t, r_s, and r_t appear in this counter-clockwise order along the spine of $\mathcal{E}$. Let G^+ be a planar supergraph of G whose vertex set is $U_b \cup U_r$ and whose edge set is $E \cup E(\sigma)$, where σ is a cycle that traverses all the vertices of U_b (and, thus, also of U_r) consecutively. A cycle σ exhibiting the above properties is a *connector* of G. By [5, Lemma 2.1] and the definition of connector, we have the following.

▶ **Lemma 23.** *A bipartite graph $G = (U_b, U_r, E)$ with distinct vertices $b_s, b_t \in U_b$ and $r_s, r_t \in U_r$ is a positive instance of* B2PBE-PE *iff it admits a connector σ in which b_t and r_t immediately precede r_s and b_s, respectively, counter-clockwise along σ.*

Next, we sketch a reduction from B2PBE-PE, which is NP-complete even for 2*-connected* graphs, to CPLS-3. As B2PBE-PE remains NP-complete even for instances with a *fixed embedding* (see [5] for details), the reduction also works if the target problem is CPLSF-3.

▶ **Lemma 24.** B2PBE-PE $\leq_m^P$ CPLS-3.



(a) Gadget $\mathcal{Q}$ **(b)** Saturation of $\mathcal{Q}$ **(c)** Gadget $\mathcal{P}$ **(d)** Saturation of $\mathcal{P}$

Figure 5 Illustrations for the gadget $\mathcal{Q}$ and $\mathcal{P}$.

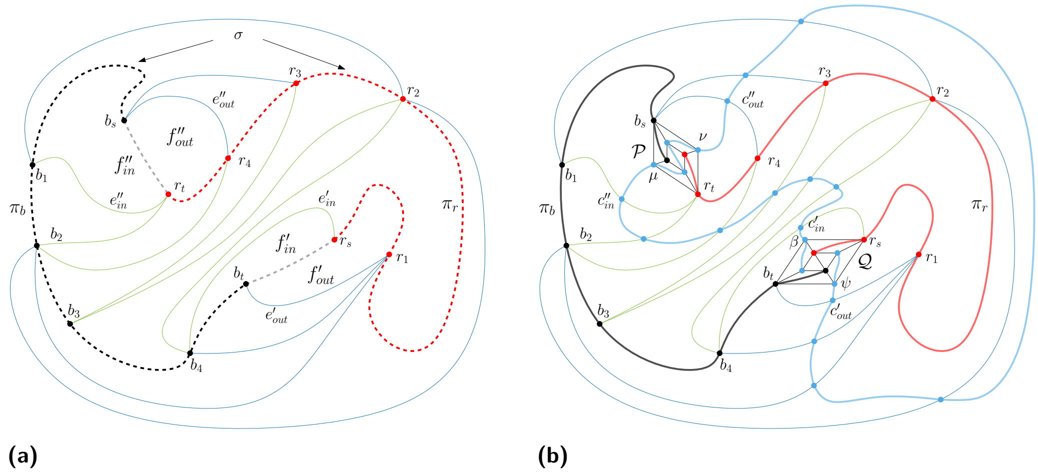


Figure 6 Illustrations for the proof of Lemma 24. (a) A planar embedding $\mathcal{E}_\sigma$ of a bipartite graph G together with a connector σ of G. The edges of σ are dashed. (b) A planar embedding of the linear saturation of the underlying graph of the independent c-graph $\mathcal{C}$, obtained from $\mathcal{E}_\sigma$.

Proof Sketch. In our reduction from B2PBE-PE to CPLS-3, we will use the *originating gadget* $\mathcal{Q}$ in Fig. 5a and the *traversing gadget* $\mathcal{P}$ in Fig. 5c. These are independent c-graphs with three clusters: red, blue, and black. Let W be a graph with 4 labeled vertices b_s, b_t, r_b, and r_t. The $\mathcal{PQ}$*-merge* of W is the graph obtained by taking the union of W, $G_\mathcal{Q}$, and $G_\mathcal{P}$, identifying the vertices b_t and r_s of W with the vertices b'_t and r'_s of $G_\mathcal{Q}$, respectively, and identifying the vertices b_s and r_t of W with the vertices b'_s and r'_t of $G_\mathcal{P}$, respectively.

Given a connected bipartite planar graph $G = (U_b, U_r, E)$ and an ordered quadruple $(b_s, b_t, r_s, r_t) \in U_b \times U_b \times U_r \times U_r$, we construct an independent c-graph $\mathcal{C} = (H, \{V_b, V_r, V_c\})$ with three clusters (red, black, and blue) as follows. First, we initialize the underlying graph H of $\mathcal{C}$ to be the $\mathcal{PQ}$-merge of G (which is well-defined, since G contains the 4 distinct vertices b_s, b_t, r_s, and r_t.). Also, we initialize $V_b = U_b \cup \{b^*, b^\diamond\}$, $V_r = U_r \cup \{r^*, r^\diamond\}$, and $V_c = \{\alpha, \beta, \lambda, \mu, \nu, \chi, \psi, \omega\}$. Second, we subdivide, in H, each edge e of $E(G)$ with a dummy vertex c_e and assign the vertex c_e to V_c.

Clearly, the above reduction can be carried out in polynomial time in the size of G. In the Full Version, we show that $\mathcal{C}$ is a positive instance of CPLS-3 if and only if (G, b_s, b_t, r_s, r_t) is a positive instance of B2PBE-PE. The crux of the proof relies on the property that, in *any* planar embedding of a linear saturation of the underlying graph of $\mathcal{C}$, the end-vertices of the path saturating the red cluster must be r^* and $r^\diamond$, the end-vertices of the path saturating the blue cluster must be α and ω, and the end-vertices of the path saturating the black cluster must be b^* and $b^\diamond$. This allows us to turn an embedding of G together with its connector (Fig. 6a) into a linear saturation of the underlying graph of $\mathcal{C}$ (Fig. 6b), and vice versa. ◀

Lemma 24 and the fact that CPLS and CPLSF clearly lie in NP imply the main result of the section, which is summarized in the following and rules out the existence of FPT algorithms for CPLS and CPLSF parameterized by the number of clusters, unless P = NP.

▶ **Theorem 25.** CPLS *and* CPLSF *are* NP*-complete even when restricted to instances with at most three clusters.*

6 Conclusions

This paper established upper and lower bounds that significantly expand our understanding of the limits of tractability for finding linear saturators in the context of clustered planarity.

We remark that, prior to this research, the problem was not known to be NP-complete for instances with $\mathcal{O}(1)$ clusters. Our NP-hardness result for instances of CPLS with three clusters narrows the complexity gap to its extreme (while also solving the open problem posed in [3, OP 4.3] about the complexity of Clique Planarity for instances with a bounded number of clusters), and animates the interest for the remaining two cluster case.

References

1 Greg Aloupis, Luis Barba, Paz Carmi, Vida Dujmovic, Fabrizio Frati, and Pat Morin. Compatible connectivity augmentation of planar disconnected graphs. *Discret. Comput. Geom.*, 54(2):459–480, 2015. doi:10.1007/S00454-015-9716-8.

2 Patrizio Angelini and Giordano Da Lozzo. Clustered planarity with pipes. *Algorithmica*, 81(6):2484–2526, 2019. doi:10.1007/S00453-018-00541-W.

3 Patrizio Angelini and Giordano Da Lozzo. Beyond clustered planar graphs. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs, Communications of NII Shonan Meetings*, pages 211–235. Springer, 2020. doi:10.1007/978-981-15-6533-5_12.

4 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Intersection-link representations of graphs. *J. Graph Algorithms Appl.*, 21(4):731–755, 2017. doi:10.7155/JGAA.00437.

5 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Maurizio Patrignani. 2-Level quasi-planarity or how caterpillars climb (SPQR-)trees. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2779–2798. SIAM, 2021. doi:10.1137/1.9781611976465.165.

6 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli. Relaxing the constraints of clustered planarity. *Comput. Geom.*, 48(2):42–75, 2015. doi:10.1016/J.COMGEO.2014.08.001.

7 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Vincenzo Roselli. The importance of being proper: (in clustered-level planarity and T-level planarity). *Theor. Comput. Sci.*, 571:1–9, 2015. doi:10.1016/J.TCS.2014.12.019.

8 Thomas Bläsius, Simon D. Fink, and Ignaz Rutter. Synchronized planarity with applications to constrained planarity problems. *ACM Trans. Algorithms*, 19(4):34:1–34:23, 2023. doi:10.1145/3607474.

9 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theor. Comput. Sci.*, 609:306–315, 2016. doi:10.1016/J.TCS.2015.10.011.

10 Ulrik Brandes and Jürgen Lerner. Visual analysis of controversy in user-generated encyclopedias. *Inf. Vis.*, 7(1):34–48, 2008. doi:10.1057/PALGRAVE.IVS.9500171.

11 Markus Chimani, Giuseppe Di Battista, Fabrizio Frati, and Karsten Klein. Advances on testing c-planarity of embedded flat clustered graphs. *Int. J. Found. Comput. Sci.*, 30(2):197–230, 2019. doi:10.1142/S0129054119500011.

12 Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Christian Wolf. Inserting a vertex into a planar graph. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 375–383. SIAM, 2009. doi:10.1137/1.9781611973068.42.

13 Markus Chimani and Petr Hlinený. Inserting multiple edges into a planar graph. In Sándor P. Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, volume 51 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.SOCG.2016.30.

14 Markus Chimani and Karsten Klein. Shrinking the search space for clustered planarity. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, volume 7704 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2012. doi:10.1007/978-3-642-36763-2_9.

15 Pier Francesco Cortese and Giuseppe Di Battista. Clustered planarity. In Joseph S. B. Mitchell and Günter Rote, editors, *Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy, June 6-8, 2005*, pages 32–34. ACM, 2005. doi:10.1145/1064092.1064093.

16 Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. C-planarity of c-connected clustered graphs. *J. Graph Algorithms Appl.*, 12(2):225–262, 2008. doi:10.7155/JGAA.00165.

17 Pier Francesco Cortese and Maurizio Patrignani. Clustered planarity = flat clustered planarity. In Therese Biedl and Andreas Kerren, editors, *Graph Drawing and Network Visualization - 26th International Symposium, GD 2018, Barcelona, Spain, September 26-28, 2018, Proceedings*, volume 11282 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2018. doi:10.1007/978-3-030-04414-5_2.

18 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. Subexponential-time and FPT algorithms for embedded flat clustered planarity. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2018. doi:10.1007/978-3-030-00256-5_10.

19 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. C-planarity testing of embedded clustered graphs with bounded dual carving-width. *Algorithmica*, 83(8):2471–2502, 2021. doi:10.1007/S00453-021-00839-2.

20 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

21 Giuseppe Di Battista and Fabrizio Frati. Efficient c-planarity testing for embedded flat clustered graphs with small faces. *J. Graph Algorithms Appl.*, 13(3):349–378, 2009. doi:10.7155/JGAA.00191.

22 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

23 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In Paul G. Spirakis, editor, *Algorithms - ESA '95, Third Annual European Symposium, Corfu, Greece, September 25-27, 1995, Proceedings*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1995. doi:10.1007/3-540-60313-1_145.

24 Sergej Fialko and Petra Mutzel. A new approximation algorithm for the planar augmentation problem. In Howard J. Karloff, editor, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, pages 260–269. ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314714.

25 F.V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. URL: https://books.google.at/books?id=s1N-DwAAQBAJ.

26 Radoslav Fulek, Jan Kyncl, Igor Malinovic, and Dömötör Pálvölgyi. Clustered planarity testing revisited. *Electron. J. Comb.*, 22(4):4, 2015. doi:10.37236/5002.

27 Radoslav Fulek and Csaba D. Tóth. Atomic embeddability, clustered planarity, and thickenability. *J. ACM*, 69(2):13:1–13:34, 2022. doi:10.1145/3502264.

28 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, Meirav Zehavi, and Liana Khazaliya. New frontiers of parameterized complexity in graph drawing (Dagstuhl Seminar 23162). *Dagstuhl Reports*, 13(4):58–97, 2023. doi:10.4230/DAGREP.13.4.58.

29 Michael T. Goodrich, George S. Lueker, and Jonathan Z. Sun. C-planarity of extrovert clustered graphs. In Patrick Healy and Nikola S. Nikolov, editors, *Graph Drawing, 13th*

International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005, Revised Papers, volume 3843 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2005. doi:10.1007/11618058_20.

30 Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel, Merijam Percan, and René Weiskircher. Advances in c-planarity testing of clustered graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002, Irvine, CA, USA, August 26-28, 2002, Revised Papers*, volume 2528 of *Lecture Notes in Computer Science*, pages 220–235. Springer, 2002. doi:10.1007/3-540-36151-0_21.

31 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

32 Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. Clustered planarity: Embedded clustered graphs with two-component clusters. In Ioannis G. Tollis and Maurizio Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, volume 5417 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2008. doi:10.1007/978-3-642-00219-9_13.

33 Vít Jelínek, Ondrej Suchý, Marek Tesar, and Tomás Vyskocil. Clustered planarity: Clusters with few outgoing edges. In Ioannis G. Tollis and Maurizio Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, volume 5417 of *Lecture Notes in Computer Science*, pages 102–113. Springer, 2008. doi:10.1007/978-3-642-00219-9_11.

34 Eva Jelínková, Jan Kára, Jan Kratochvíl, Martin Pergel, Ondrej Suchý, and Tomás Vyskocil. Clustered planarity: Small clusters in cycles and eulerian graphs. *J. Graph Algorithms Appl.*, 13(3):379–422, 2009. doi:10.7155/JGAA.00192.

35 Tomihisa Kamada and Satoru Kawai. A general framework for visualizing abstract objects and relations. *ACM Trans. Graph.*, 10(1):1–39, 1991. doi:10.1145/99902.99903.

36 Goos Kant and Hans L. Bodlaender. Planar graph augmentation problems (extended abstract). In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 2nd Workshop WADS '91, Ottawa, Canada, August 14-16, 1991, Proceedings*, volume 519 of *Lecture Notes in Computer Science*, pages 286–298. Springer, 1991. doi:10.1007/BFB0028270.

37 Thomas Lengauer. Hierarchical planarity testing algorithms. *J. ACM*, 36(3):474–509, 1989. doi:10.1145/65950.65952.

38 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

39 Giordano Da Lozzo, Robert Ganian, Siddharth Gupta, Bojan Mohar, Sebastian Ordyniak, and Meirav Zehavi. Exact algorithms for clustered planarity with linear saturators, 2024. arXiv:2409.19410.

40 Oliver Niggemann. *Visual data mining of graph based data*. PhD thesis, University of Paderborn, Germany, 2001. URL: http://ubdata.uni-paderborn.de/ediss/17/2001/niggeman/disserta.pdf.

41 Renato Paiva, Genaína Nunes Rodrigues, Rodrigo Bonifácio, and Marcelo Ladeira. Exploring the combination of software visualization and data clustering in the software architecture recovery process. In Sascha Ossowski, editor, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 1309–1314. ACM, 2016. doi:10.1145/2851613.2851765.

42 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 706–715. ACM, 1994. doi:10.1145/195058.195439.

43 Jamie Sneddon and C. Paul Bonnington. A note on obstructions to clustered planarity. *Electron. J. Comb.*, 18(1), 2011. doi:10.37236/646.

The Complexity of Geodesic Spanners Using Steiner Points

Sarita de Berg
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tim Ophelders
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Irene Parada
Department of Mathematics, Universitat Politècnica de Catalunya, Barcelona, Spain

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Jules Wulms
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

A geometric t-spanner $\mathcal{G}$ on a set S of n point sites in a metric space P is a subgraph of the complete graph on S such that for every pair of sites p, q the distance in $\mathcal{G}$ is a most t times the distance $d(p, q)$ in P. We call a connection between two sites a *link*. In some settings, such as when P is a simple polygon with m vertices and a link is a shortest path in P, links can consist of $\Theta(m)$ segments and thus have non-constant complexity. The spanner complexity is a measure of how compact a spanner is, which is equal to the sum of the complexities of all links in the spanner. In this paper, we study what happens if we are allowed to introduce k Steiner points to reduce the spanner complexity. We study such Steiner spanners in simple polygons, polygonal domains, and edge-weighted trees.

Surprisingly, we show that Steiner points have only limited utility. For a spanner that uses k Steiner points, we provide an $\Omega(nm/k)$ lower bound on the worst-case complexity of any $(3-\varepsilon)$-spanner, and an $\Omega(mn^{1/(t+1)}/k^{1/(t+1)})$ lower bound on the worst-case complexity of any $(t-\varepsilon)$-spanner, for any constant $\varepsilon \in (0, 1)$ and integer constant $t \geq 2$. These lower bounds hold in all settings. Additionally, we show NP-hardness for the problem of deciding whether a set of sites in a polygonal domain admits a 3-spanner with a given maximum complexity using k Steiner points.

On the positive side, for trees we show how to build a $2t$-spanner that uses k Steiner points of complexity $O(mn^{1/t}/k^{1/t} + n\log(n/k))$, for any integer $t \geq 1$. We generalize this result to forests, and apply it to obtain a $2\sqrt{2}t$-spanner in a simple polygon with total complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$. When a link in the spanner can be any path between two sites, we show how to improve the spanning ratio in a simple polygon to $(2k+\varepsilon)$, for any constant $\varepsilon \in (0, 2k)$, and how to build a $6t$-spanner in a polygonal domain with the same complexity.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases spanner, simple polygon, polygonal domain, geodesic distance, complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.25

Related Version *Full Version*: https://arxiv.org/abs/2402.12110 [13]

Funding *Tim Ophelders*: partially supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.
Irene Parada: I. P. is a Serra Húnter Fellow. Partially supported by grant 2021UPC-MS-67392 funded by the Spanish Ministry of Universities and the European Union (NextGenerationEU) and by grant PID2019-104129GB-I00 funded by MICIU/AEI/10.13039/501100011033.

Acknowledgements Work initiated at the 2023 AGA Workshop in Otterlo. We would like to thank Wouter Meulemans for his contributions in the initial phase of the project.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 25; pp. 25:1–25:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a set S of n point *sites* in a metric space P. In applications such as (wireless) network design [3], regression analysis [19], vehicle routing [12, 26], and constructing TSP tours [6], it is desirable to have a compact network that accurately captures the distances between the sites in S. Spanners provide such a representation. Formally, a *geometric t-spanner* $\mathcal{G}$ is a subgraph of the complete graph on S, so that for every pair of sites p, q the distance $d_{\mathcal{G}}(p, q)$ in $\mathcal{G}$ is at most t times the distance $d(p, q)$ in P [24]. The quality of a spanner can be expressed in terms of the *spanning ratio* t and a term to measure how "compact" it is. Typical examples are the *size* of the spanner, that is, the number of edges of $\mathcal{G}$, its weight (the sum of the edge lengths), or its diameter. Such spanners are well studied [4, 8, 10, 18]. For example, for point sites in $\mathbb{R}^d$ and any constant $\varepsilon > 0$ one can construct a $(1 + \varepsilon)$-spanner of size $O(n/\varepsilon^{d-1})$ [25]. Similar results exist for more general spaces [9, 20, 21]. Furthermore, there are various spanners with other desirable spanner properties such as low maximum degree, or fault-tolerance [7, 22, 23, 25].

When the sites represent physical locations, there are often other objects (e.g. buildings, lakes, roads, mountains) that influence the shortest path between the sites. In such settings, we need to explicitly incorporate the environment. We consider the case where this environment is modeled by a polygon P with m vertices, and possibly containing holes. The distance between two points $p, q \in P$ is then given by their *geodesic distance*: the length of a shortest path between p and q that is fully contained in P. This setting has been considered before. For example, Abam, Adeli, Homapou, and Asadollahpoor [1] present a $(\sqrt{10} + \varepsilon)$-spanner of size $O(n \log^2 n)$ when P is a simple polygon, and a $(5+\varepsilon)$-spanner of size $O(n\sqrt{h} \log^2 n)$ when the polygon has $h > 1$ holes. Abam, de Berg, and Seraji [2] even obtain a $(2 + \varepsilon)$-spanner of size $O(n \log n)$ when P is actually a terrain. To avoid confusion between the edges of P and the edges of $\mathcal{G}$, we will from hereon use the term *links* to refer to the edges of $\mathcal{G}$.

As argued by de Berg, van Kreveld, and Staals [15], each link in a geodesic spanner may correspond to a shortest path containing $\Omega(m)$ polygon vertices. Therefore, the *spanner complexity*, defined as the total number of line segments that make up all links in the spanner, more appropriate measures how compact a geodesic spanner is. In this definition, a line segment that appears in multiple links is counted multiple times: once for each link it appears in. The above spanners of [1, 2] all have worst-case complexity $\Omega(mn)$, hence de Berg, van Kreveld, and Staals present an algorithm to construct a $2\sqrt{2}t$-spanner in a simple polygon with complexity $O(mn^{1/t} + n \log^2 n)$, for any integer $t \geq 1$. By relaxing the restriction of links being shortest paths to any path between two sites, they obtain, for any constant $\varepsilon \in (0, 2t)$, a *relaxed* geodesic $(2t + \varepsilon)$-spanner in a simple polygon, or a relaxed geodesic $6t$-spanner in a polygon with holes, of the same complexity. These complexity bounds are still relatively high. De Berg, van Kreveld, and Staals [15] also show that these results are almost tight. In particular, for sites in a simple polygon, any geodesic $(3 - \varepsilon)$-spanner has worst-case complexity $\Omega(nm)$, and for any constant $\varepsilon \in (0, 1)$ and integer constant $t \geq 2$, a $(t - \varepsilon)$-spanner has worst-case complexity $\Omega(mn^{1/(t-1)} + n)$.

Problem Statement. A very natural question is then if we can reduce the total complexity of a geodesic spanner by allowing *Steiner points*. That is, by adding an additional set $\mathcal{S}$ of k vertices in $\mathcal{G}$, each one corresponding to a (Steiner) point in P. For the original sites $p, q \in S$ we still require that their distance in $\mathcal{G}$ is at most t times their distance in P, but the graph distance from a Steiner point $p' \in \mathcal{S}$ to any other site is unrestrained. Allowing for such Steiner points has proven to be useful in reducing the weight [5, 17] and size [22]

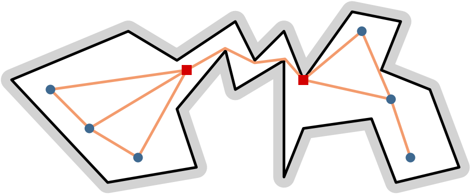


Figure 1 A spanner in a simple polygon that uses two Steiner points (red squares). By adding the two Steiner points, the spanner has a small spanning ratio and low complexity, as we no longer need multiple links that pass through the middle section of P.

of spanners. In our setting, it allows us to create additional "junction" vertices, thereby allowing us to share high-complexity subpaths. See Figure 1 for an illustration. Indeed, if we are allowed to turn every polygon vertex into a Steiner point, Clarkson [11] shows that, for any $\varepsilon > 0$, we can obtain a $(1+\varepsilon)$-spanner of complexity $O((n+m)/\varepsilon)$. However, the number of polygon vertices m may be much larger than the number of Steiner points we can afford. Hence, we focus on the scenario in which the number of Steiner points k is (much) smaller than m and n.

Our Contributions. Surprisingly, we show that in this setting, Steiner points have only limited utility. In some cases, even a single Steiner point allows us to improve the complexity by a linear factor. However, we show that such improvements are not possible in general. First of all, we show that computing a minimum cardinality set of Steiner points for sites in a polygonal domain that allow for a 3-spanner of a certain complexity is NP-hard. Moreover, we show that there is a set of n sites in a simple polygon with $m = \Omega(n)$ vertices for which any $(2-\varepsilon)$-spanner (with $k < n/2$ Steiner points) has complexity $\Omega(mn^2/k^2)$. Similarly, we give a $\Omega(mn/k)$ and $\Omega(mn^{1/(1+t)}/k^{1/(1+t)})$ lower bound on the complexity of a $(3-\varepsilon)$- and $(t-\varepsilon)$-spanner with k Steiner points. Hence, these results dash our hopes for a near linear complexity spanner with "few" Steiner points and constant spanning ratio.

These lower bounds actually hold in a more restricted setting. Namely, when the metric space is simply an edge-weighted tree that has m vertices, and the n sites are all placed in leaves of the tree. In this setting, we show that we can efficiently construct a spanner whose complexity is relatively close to optimal. In particular, our algorithm constructs a $2t$-spanner of complexity $O(mn^{1/t}/k^{1/t} + n\log(n/k))$. The main idea is to partition the tree into k subtrees of roughly equal size, construct a $2t$-spanner without Steiner points on each subtree, and connect the spanners of adjacent trees using Steiner points. The key challenge that we tackle, and one of the main novelties of the paper, is to make sure that each subtree contains only a constant number of Steiner points. We carefully argue that such a partition exists, and that we can efficiently construct it. Constructing the spanner takes $O(n\log(n/k) + m + K)$ time, where K is the output complexity. This output complexity is either the size of the spanner ($O(n\log(n/k))$), in case we only wish to report the endpoints of the links, or the complexity, in case we wish to explicitly report the shortest paths making up the links. An extension of this algorithm allows us to deal with a forest as well.

This algorithm for constructing a spanner on an edge-weighted tree turns out to be the crucial ingredient for constructing low-complexity spanners for point sites in polygons. In particular, given a set of sites in a simple polygon P, we use some of the techniques developed by de Berg, van Kreveld, and Staals [15] to build a set of trees whose leaves are the sites, and in which the distances in the trees are similar to the distances in the polygon. We then construct a $2t$-spanner with k Steiner points on this forest of trees using the above algorithm, and argue that this actually results into a $2\sqrt{2}t$-spanner with respect to the distances in the polygon. The main challenge here is to argue that the links used still have low complexity, even when they are now embedded in the polygon. We prove that the spanner (with respect to the polygon) has complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$, and can be constructed in time $O(n\log^2 n + m\log n + K)$. If we allow a link in the spanner to be any path between two sites (or Steiner points), then we obtain for any constant $\varepsilon \in (0, 2k)$ a relaxed $(2t+\varepsilon)$-spanner of the same complexity. For $k = O(1)$ our spanners thus match the results of de Berg, van Kreveld, and Staals [15]. Finally, we extend these results to polygonal domains, where we construct a similar complexity relaxed $6t$-spanner in $O(n\log^2 n + m\log n\log m + K)$ time.

Organization. We start with our results on edge-weighted trees in Section 2. To get a feel for the problem, we first establish lower bounds on the spanner complexity in Section 2.1. In Section 2.2 we present the algorithm for efficiently constructing a low complexity $2t$-spanner, in the full version [13] we extend it to a forest. In Section 3, we show how to use these results to obtain a $2\sqrt{2}t$-spanner for sites in a simple polygon P. In Section 4 we further extend our algorithms to the most general case in which P may even have holes. In the full version of the paper [13] we show that computing a minimum cardinality set of Steiner points with which we can simultaneously achieve a particular spanning ratio and maximum complexity is NP-hard. In Section 5 we pose some remaining open questions. Omitted proofs can be found in the full version [13].

2 Steiner spanners for trees

In this section, we consider spanners on an edge-weighted rooted tree T. We allow only positive weights. The goal is to construct a t-spanner on the leaves of the tree that uses k Steiner points, i.e. the set of sites S is the set of leaves. We denote by n the number of leaves and by m the number of vertices in T. The complexity of a link between two sites (or Steiner points) $p, q \in T$ is the number of edges in the shortest path $\pi(p, q)$, and the distance $d(p, q)$ is equal to the sum of the weights on this (unique) path. We denote by $T(v)$ the subtree of T rooted at vertex v. For an edge $e \in T$ with upper endpoint v_1 (endpoint closest to the root) and lower endpoint v_2, we denote by $T(e) := T(v_2) \cup \{e\}$ the subtree of T rooted at v_1.

The Steiner points are not restricted to the vertices of T, but can lie anywhere on the tree. To be precise, for any $\delta \in (0, 1)$ a Steiner point s can be placed on an edge (u, v) of weight w. This edge is then replaced by two edges (u, s) and (s, v) of weight δw and $(1-\delta)w$. Observe that this increases the complexity of a spanner on T by at most a constant factor as long as there are at most a constant number of Steiner points placed on a single edge. The next lemma states that it is indeed never useful to place more than one Steiner point on the interior of an edge.

▶ **Lemma 1.** *If a t-spanner $\mathcal{G}$ of a tree T has more than one Steiner point on the interior of an edge $e = (u, v)$, then we can modify $\mathcal{G}$ to obtain a t-spanner $\mathcal{G}'$ that has no Steiner points on the interior of e without increasing the complexity and number of Steiner points.*

Proof. Let $\mathcal{S}$ denote the set of Steiner points of $\mathcal{G}$ and let $\mathcal{S}(e) \subseteq \mathcal{S}$ the subset of Steiner points that lie on e. We assume that each Steiner point is used by a path $\pi_{\mathcal{G}}(p, q)$ for some sites p, q, otherwise we can simply remove it. We define the set of Steiner points of $\mathcal{G}'$ as $\mathcal{S}' = (\mathcal{S} \setminus \mathcal{S}(e)) \cup \{u, v\}$. Observe that $|\mathcal{S}'| \leq |\mathcal{S}|$. To obtain $\mathcal{G}'$, we replace each link (p, s) with $s \notin \mathcal{S}'$ by (p, u) if (p, s) intersects u and by (p, v) if (p, s) intersects v. Links between Steiner points on e are simply removed. Finally, we add the link (u, v) to $\mathcal{G}'$.

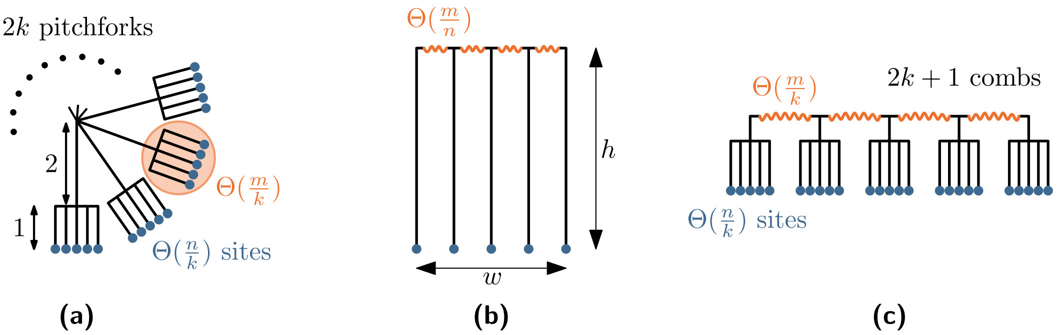


Figure 2 **(a)** Our construction for an $\Omega(mn^2/k^2)$ lower bound on the complexity of any $(2-\varepsilon)$-spanner. **(b)** A more detailed version of the comb of a pitchfork highlighted in the orange disk, which is also used for our $\Omega(mn^{1/(t+1)}/k^{1/(t+1)})$ lower bound on the complexity of any $(t-\varepsilon)$-spanner. **(c)** Our construction for an $\Omega(nm/k)$ lower bound on the complexity of any $(3-\varepsilon)$-spanner.

We first argue that the spanning ratio of $\mathcal{G}'$ is as most the spanning ratio of $\mathcal{G}$. Consider a path between two sites p, q in $\mathcal{G}$. If this path still exists in $\mathcal{G}'$, then $d_{\mathcal{G}}(p,q) = d_{\mathcal{G}'}(p,q)$. If not, then the path must visit e. Let (p_1, s_1) and (p_2, s_2) denote the first and last link in the path that connect to a Steiner point in the interior of e (possibly $s_1 = s_2$). If $\pi(p,q)$ does not intersect the open edge e, then these links are replaced by (p_1, u) and (p_2, u) (or symmetrically by (p_1, v) and (p_2, v)) in $\mathcal{G}'$. This gives a path in $\mathcal{G}'$ via u such that $d_{\mathcal{G}'}(p,q) < d_{\mathcal{G}}(p,q)$. If $\pi(p,q)$ does intersect e, i.e. p and q lie on different sides of e, then, without loss of generality, the links (p_1, s_1) and (p_2, s_2) are replaced by (p_1, u), (u, v), and (p_2, v). Again, this gives a path in $\mathcal{G}'$ such that $d_{\mathcal{G}'}(p,q) \leq d_{\mathcal{G}}(p,q)$.

Finally, what remains is to argue that the complexity of the spanner does not increase. Each link that we replace intersects either u or v, thus replacing this link by a link up to u or v reduces the complexity by one. Because each Steiner point on e occurs on at least one path between sites in $\mathcal{G}$, we replace at least two links. This decreases the total complexity by at least two, while including the edge (u, v) increases the complexity by only one. ◀

▶ **Corollary 2.** *Any spanner $\mathcal{G}$ on a tree T can be modified without increasing the spanning ratio and complexity such that no edge contains more than one Steiner point in its interior.*

2.1 Complexity lower bounds

In this section, we provide several lower bounds on the worst-case complexity of any $(t-\varepsilon)$-spanner that uses k Steiner points, where t is an integer constant and $\varepsilon \in (0,1)$. When Steiner points are not allowed, any $(2-\varepsilon)$-spanner in a simple polygon requires $\Omega(n^2)$ edges [1] and $\Omega(mn^2)$ complexity. If we allow a larger spanning ratio, say $(3-\varepsilon)$ or even $(t-\varepsilon)$, the worst-case complexity becomes $\Omega(mn)$ or $\Omega(mn^{1/(t-1)})$, respectively [15]. As the polygons used for these lower bounds are very tree-like, these bounds also hold in our tree setting. Next, we show how much each of these lower bounds is affected by the use of k Steiner points.

▶ **Lemma 3.** *For any constant $\varepsilon \in (0,1)$, there exists an edge-weighted tree T for which any $(2-\varepsilon)$-spanner using $k < n/2$ Steiner points has complexity $\Omega(mn^2/k^2)$.*

Proof sketch. The tree in Figure 2(a) is used to show this bound. Each of the k pitchforks that does not contain a Steiner point contributes $\Omega(mn^2/k^3)$ complexity to the spanner. ◀

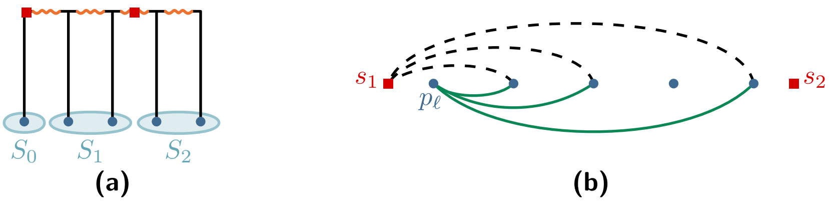


Figure 3 (a) The sets S_i defined by the two (red square) Steiner points. **(b)** For any spanner on S_1, every link to a Steiner point can be replaced by a link of smaller complexity, while increasing the spanning ratio by at most one. Here, the dashed links can be replaced by the green links.

▶ **Lemma 4.** *For any constant $\varepsilon \in (0, 1)$, there exists an edge-weighted tree T for which any $(3 - \varepsilon)$-spanner using $k < n/2$ Steiner points has complexity $\Omega(mn/k)$.*

Proof sketch. The tree in Figure 2(c) is used to show this lower bound. ◀

▶ **Lemma 5.** *For any constant $\varepsilon \in (0, 1)$ and integer constant $t \geq 2$, there exists an edge-weighted tree T for which any $(t - \varepsilon)$-spanner using $k < n$ Steiner points has complexity $\Omega(mn^{1/(t+1)}/k^{1/(t+1)})$.*

Before we prove Lemma 5, we first discuss a related result in a simpler metric space. Let ϑ_n be the 1-dimensional Euclidean metric space with n points $v_1, \dots, v_n$ on the x-axis at $1, 2, \dots, n$. A link (v_i, v_j) has complexity $|i - j|$. Dinitz, Elkin, and Solomon [16] give a lower bound on the total complexity of any spanning subgraph of ϑ_n, given that the link-radius is at most ρ. The link-radius (called hop-radius in [16]) $\rho(G, r)$ of a graph G with respect to a root r is defined as the maximum number of links needed to reach any vertex in G from r. The link-radius of G is then $\min_{r \in V} \rho(G, r)$. The link-radius is bounded by the link-diameter, which is the minimum number of links that allow reachability between any two vertices.

▶ **Lemma 6** (Dinitz et al. [16]). *For any sufficiently large integer n and positive integer $\rho < \log n$, any spanning subgraph of ϑ_n with link-radius at most ρ has complexity $\Omega(\rho \cdot n^{1+1/\rho})$.*

Proof of Lemma 5. Consider the tree construction illustrated in Figure 2(b). This edge-weighted tree T has the shape of a comb of width w and height h with n teeth separated by corridors of complexity $M = \Theta(m/n)$ each. Each leaf at the bottom of a comb tooth is a site.

Any spanning subgraph of ϑ_n of complexity C and link-diameter ρ is in one-to-one correspondence with a $(\rho + 1 - \varepsilon)$-spanner of complexity $C \cdot m/n$ in T [15]. Lemma 6 then implies that any $(t - \varepsilon)$-spanner has worst-case complexity $\Omega(mn^{1/(t-1)})$.

When a set $\mathcal{S}$ of k Steiner points is introduced, we consider the at most $k + 1$ sets $S_0, \dots, S_k$ of consecutive sites that have no Steiner point between them; see Figure 3(a). We can replace any link (p, q) where $p, q \in S \cup \mathcal{S}$ and $\pi(p, q)$ intersects a Steiner point s by the links (p, s) and (s, q). Corollary 2 implies that this increases the complexity by only a constant factor. From now on, we thus assume there are no such links. We claim the following for any $(t - \varepsilon)$-spanner $\mathcal{G}$ on $S = S_0 \cup \dots \cup S_k$.

▷ Claim 7. Let C_i be the complexity of the subgraph of $\mathcal{G}$ induced by S_i and the at most two Steiner points s_ℓ and s_r bounding S_i from the left and right, respectively. Then, we can construct a $(t + 2 - \varepsilon)$-spanner $\mathcal{G}'_i$ on S_i that has complexity at most C_i.

Proof of Claim. Let p_ℓ and p_r denote the leftmost and rightmost site in S_i. We replace each link (p, s_ℓ) (or (p, s_r)), $p \in S_i$, by the link (p, p_ℓ) (resp. (p, p_r)). If there is a link (s_ℓ, s_r), it is replaced by (p_ℓ, p_r). Any path in $\mathcal{G}$ between $p, q \in S_i$ that visits either s_ℓ and/or s_r

corresponds to a path via p_r and/or p_ℓ in $\mathcal{G}'_i$. The length of the path increases by $2h$ when visiting p_r or p_ℓ, so by at most $4h$ when visiting both. As $d(p,q) \geq 2h$, the spanning ratio increases by at most two. ◁

These changes in the spanner only decrease the complexity of the subspanner on S_i. Notice also that if we apply them to each of the sets S_i, each link of $\mathcal{G}$ is changed by only one of the subspanners $\mathcal{G}'_i$. Thus, we consider the minimum complexity of any $(t+2-\varepsilon)$-spanner on these sites. By applying Lemma 6, we find that the worst-case complexity of any $(t+2-\varepsilon)$-spanner on these $|S_i|$ sites is $\Omega(m/n \cdot |S_i|^{1+1/(t+1)})$. The complexity of $\mathcal{G}$ is at least the sum of the complexities of these $\mathcal{G}'_i$ spanners over all S_i, so $\frac{m}{n}\sum_{i=0}^{k}\Omega\left(|S_i|^{1+1/(t+1)}\right)$, where $\sum_{i=0}^{k}|S_i| = \Theta(n)$. Using a logarithmic transformation and induction, we see that this sum is minimized when $|S_i| = \Theta(n/k)$ for all $i \in 0,\ldots,k$. So,

$$\frac{m}{n}\sum_{i=0}^{k}\Omega\left(|S_i|^{1+1/(t+1)}\right) \geq \frac{m}{n}\sum_{i=0}^{k}\Omega\left((n/k)^{1+1/(t+1)}\right) = \Omega\left(mn^{1/(t+1)}/k^{1/(t+1)}\right). \quad ◀$$

2.2 A low complexity Steiner spanner

In this section, we describe how to construct low complexity spanners for edge-weighted trees. The goal is to construct a $2t$-spanner of complexity $O(mn^{1/t}/k^{1/t} + n\log n)$ that uses at most k Steiner points. We first show that the spanner construction for a simple polygon of [15] can be used to obtain a low complexity spanner for a tree (without Steiner points).

▶ **Lemma 8** (de Berg et al. [15]). *For any integer $t \geq 1$, we can build a $2t$-spanner for an edge-weighted tree T of size $O(n\log n)$ and complexity $O(mn^{1/t} + n\log n)$ in $O(n\log n + m)$ time.*

Spanner construction. Given an edge-weighted tree T, to construct a Steiner spanner $\mathcal{G}$ for T, we start by partitioning the sites in k sets $S_1,\ldots S_k$ by an in-order traversal of the tree. The first $\lceil n/k\rceil$ sites encountered are in S_1, the second $\lceil n/k\rceil$ in S_2, etc. After this, the sites are reassigned into k new disjoint sets $S'_1,\ldots S'_k$. For each of these sets, we consider a subtree $T'_i \subseteq T$ whose leaves are the set S'_i. There are four properties that we desire of these sets and their subtrees.

1. The size of S'_i is $O(n/k)$.
2. The trees T'_i cover T, i.e. $\bigcup_i T'_i = T$.
3. The trees T'_i are disjoint apart from Steiner points.
4. Each tree T'_i contains only $O(1)$ Steiner points.

As we prove later, these properties ensure that we can construct a spanner on each subtree T'_i to obtain a spanner for T. We obtain such sets S'_i and the corresponding trees T'_i as follows.

We color the vertices and edges of the tree T using k colors $\{1,\ldots,k\}$ in two steps. In this coloring, an edge or vertex is allowed to have more than one color. First, for each set S_i, we color the smallest subtree that contains all sites in S_i with color i. After this step, all uncolored vertices have only uncolored incident descendant edges. Second, we color the remaining uncolored edges and vertices. These edges and their (possibly already colored) upper endpoints are colored in a bottom-up fashion. We assign each uncolored edge and its upper endpoint the color with the lowest index i that is assigned also to its lower endpoint.

After coloring T, we for $i \in \{1,\ldots,k\}$ place a Steiner point s_i at the root of tree T_i formed by all edges and vertices of color i. This may place multiple Steiner points at the same vertex. We may abuse notation, and denote by s_i the vertex occupied by Steiner point s_i.

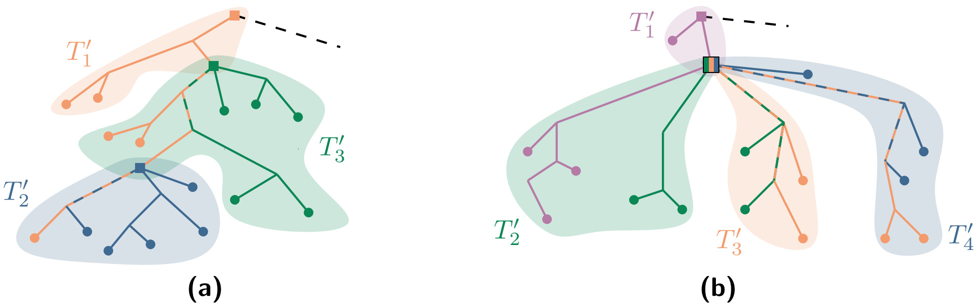


Figure 4 The tree T_i is the subtree whose edges and vertices have color i. A Steiner point (square) is placed at the root of T_i. The shaded areas show the trees T'_i. The examples show the case when the Steiner points are **(a)** at different vertices or **(b)** share a vertex.

For each Steiner point s_i, we define a subtree $T'_i \subseteq T$. The sites in T'_i will be the set S'_i. The tree T'_i is a subtree of $T(s_i)$. When s_i is the only Steiner point at the vertex, then $T'_i = T(s_i) \setminus \bigcup_j (T(s_j) \setminus \{s_j\})$ for s_j a descendant of s_i. In other words, we look at the tree rooted at s_i up to and including the next Steiner points, see Figure 4(a). When s_i is not the only Steiner point at the vertex, we include only subtrees $T(e)$ of s_i (up to the next Steiner points) that start with an edge e that has color i and no color $j > i$. See Figure 4(b). Whenever s_i has the lowest or highest index of the Steiner points at s_i, we also include all $T(e')$ that start with an edge e' of color $j < i$ or $j > i$, respectively. This generalizes the scheme for when s_i is the only Steiner point at the vertex.

By creating T'_i in this way, s_i is not a leaf of T'_i. We therefore adapt T'_i by adding an edge of weight zero between the vertex at s_i and a new leaf corresponding to s_i. On each subtree T'_i, we construct a $2t$-spanner using the algorithm of Lemma 8. These k spanners connect at the Steiner points, which we formally prove in the spanner analysis.

Analysis. To prove that $\mathcal{G}$ is indeed a low complexity $2t$-spanner for T, we first show that the four properties stated before hold for S'_i and T'_i. We often apply the following lemma, that limits the number of colors an edge can be assigned by our coloring scheme.

▶ **Lemma 9.** *An edge can have at most two colors.*

Proof. First, observe that an edge can receive more than one color only in the first step of the coloring. Suppose for contradiction that there is an edge e in T that has three colors $i < j < \ell$. Let v be the lower endpoint of e. Then there must be three sites $p_i \in S_i$, $p_j \in S_j$, $p_\ell \in S_\ell$ in $T(v)$. Because these sets are defined by an in-order traversal, p_i must appear before p_j in the traversal. Similarly, p_j appears before p_ℓ. Additionally, there must be a site $p'_j \in S_j$ in $T \setminus T(v)$, otherwise the color j would not be assigned to e. The site p'_j must appear before p_i or after p_ℓ in the traversal. In the first case, p_i must be in S_j as it appears between two sites in S_j. In the second case, we find $p_\ell \in S_j$, also giving a contradiction. ◀

We prove properties 1, 2, and 3 in the full version [13].

▶ **Lemma 10.** *There are at most five Steiner points in T'_i.*

Proof sketch. By definition, s_i is in T'_i, so we want to show that there are at most four other Steiner points in T'_i. Note that a Steiner point can occur in T'_i only if its path to s_i does not encounter any other Steiner point. In this proof sketch we show there are at most

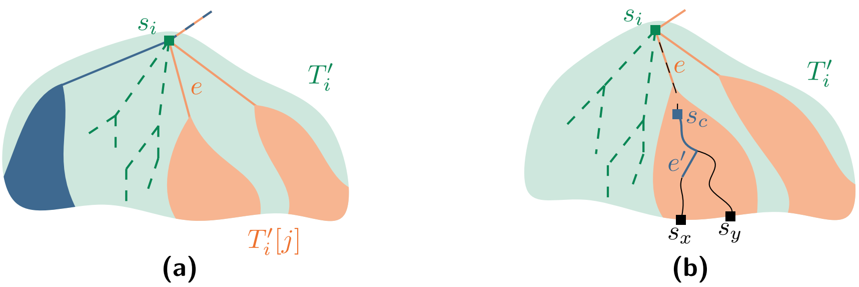


Figure 5 Notation used in Lemma 10. In $T'_i[j]$ are all subtrees that start with an edge of color j.

two Steiner points in subtrees $T(e)$ for which the edge e, which is incident to s_i in T'_i, does not have color i. In the full proof we use similar techniques to bound the number of Steiner points in subtrees for which e does have color i by two as well.

Let $T'_i[j]$ be the subtree of T'_i rooted at s_i that is the union of $T(e) \cap T'_i$ for all edges e incident to s_i of color $j \neq i$ and not of color i as well, see Figure 5(a). We argue that this subtree is non-empty for at most two colors j. Consider such an edge e. Because e does not have color i and $e \in T'_i$, it must be that s_j is above s_i in T. Thus, the parent edge of s_i in T must also be colored j. By Lemma 9, the parent edge of s_i in T can be assigned at most two colors, so $T'_i[j]$ is non-empty for at most two colors.

Next, we prove that $T'_i[j]$ contains at most one Steiner point other than s_i. We assume that $i < j$, the proof for $i > j$ is symmetric. Assume for contradiction that $T'_i[j]$ contains two Steiner points s_x and s_y, $x < y$; see Figure 5(b). As shown before, there is a site of S_j in $T \setminus T(s_i)$. As $i < j$, this implies that $i < x < y < j$. Let e' be the first edge on $\pi(s_i, s_x)$ that is not on $\pi(s_i, s_y)$, i.e. the first edge after the paths diverge. Let c be a color of e' and let v and w be the upper and lower endpoint of e'. The tree $T(w)$ does not contain any sites of S_j, as these appear in the traversal after the sites of S_x. It follows that S_c is before S_j in the in-order traversal, in other words $i < c < j$. The parent edge of s_i cannot be colored c, as a site of S_c would then appear either before a site in S_i or after a site in S_j in the in-order traversal. It follows that s_c is on the path $\pi(v, s_i)$. If $s_c \neq s_i$, this contradicts the assumption that this path does not contain a Steiner point. If $s_c = s_i$, then $i < c$ implies that the subtree starting with an edge of color j is in not T'_i, which is a contradiction. We conclude that there are at most two Steiner points in the subtrees $T'_i[j]$ in total for all $j \neq i$. ◀

We are now ready to prove that our algorithm computes a spanner with low complexity.

▶ **Lemma 11.** *The spanner $\mathcal{G}$ is a $2t$-spanner for T of size $O(n \log(n/k))$ and complexity $O(mn^{1/t}/k^{1/t} + n \log(n/k))$.*

Proof. To bound the size and complexity of the spanner, we first consider the number of leaves n_i and vertices m_i in each subtree T'_i. As n_i is equal to $|S'_i|$ plus the number of Steiner points in T'_i, properties 1 and 4 (Lemma 10) imply that $n_i = O(n/k) + 5 = O(n/k)$. Property 3 states the subtrees T'_i are disjoint apart from their shared Steiner points, so $\sum m_i = O(m)$. By Lemma 8, $\mathcal{G}$ has size $\sum_{i=1}^{k} O\left(\frac{n}{k} \log\left(\frac{n}{k}\right)\right) = O\left(n \log\left(\frac{n}{k}\right)\right)$ and complexity $\sum_{i=1}^{k} O\left(m_i \left(\frac{n}{k}\right)^{1/t} + \frac{n}{k} \log\left(\frac{n}{k}\right)\right) = O\left(\frac{mn^{1/t}}{k^{1/t}} + n \log\left(\frac{n}{k}\right)\right)$.

What remains is to show that $\mathcal{G}$ is a $2t$-spanner. Let $p, q \in S$ be two leaves in T. If $p, q \in S'_i$ for some $i \in \{1, \ldots, k\}$ then the shortest path $\pi(p, q)$ is contained within T'_i. The $2t$-spanner on T'_i implies that $d_{\mathcal{G}}(p, q) \leq 2td(p, q)$. If there is no such set S'_i that contains both sites, consider the sequence of vertices $v_1, \ldots, v_\ell$ where $\pi(p, q)$ exits some subtree T'_i. Let v, w be two consecutive vertices in this sequence. Without loss of generality, assume that $w \in T(v)$, and let s_x be the Steiner point at v for which $w \in T'_x$ (properties 2 and 3 imply s_x exists). Then the $2t$-spanner on T'_x ensures that $d_{\mathcal{G}}(v, w) \leq 2td(v, w)$. It follows that $d_{\mathcal{G}}(p, q) \leq d_{\mathcal{G}}(p, v_1) + d_{\mathcal{G}}(v_1, v_2) + \cdots + d_{\mathcal{G}}(v_\ell, q) \leq 2t(d(p, v_1) + d(v_1, v_2) + \cdots + d(v_\ell, q)) = 2td(p, q)$. ◀

▶ **Theorem 12.** *Let T be a tree with n leaves and m vertices, and $t \leq 1$ be any integer constant. For any $1 \leq k \leq n$, we can build a $2t$-spanner $\mathcal{G}$ for T using at most k Steiner points of size $O(n \log(n/k))$ and complexity $O(mn^{1/t}/k^{1/t} + n \log(n/k))$ in $O(n \log(n/k) + m + K)$ time, where K is the output size.*

A forest spanner. The tree spanner can be extended to a spanner for a forest $\mathcal{F}$. As $\mathcal{F}$ is disconnected, we cannot require all sites to have a path between them in the spanner. Instead, we say that $\mathcal{G}$ is a t-spanner for $\mathcal{F}$ if $\mathcal{G}$ is a t-spanner for every tree in $\mathcal{F}$.

▶ **Theorem 13.** *Let $\mathcal{F}$ be a forest with n leaves and m vertices, and $t \leq 1$ be any integer constant. For any $1 \leq k \leq n$, we can build a $2t$-spanner $\mathcal{G}$ for $\mathcal{F}$ using at most k Steiner points of size $O(n \log(n/k))$ and complexity $O(mn^{1/t}/k^{1/t} + n \log(n/k))$ in $O(n \log(n/k) + m + K)$ time, where K is the output size.*

3 Steiner spanners in simple polygons

We consider the problem of computing a t-spanner using k Steiner points for a set of n point sites in a simple polygon P with m vertices. We measure the distance between two points in p, q in P by their geodesic distance, i.e. the length of the shortest path $\pi(p, q)$ fully contained within P. A link (p, q) in the spanner is the shortest path $\pi(p, q)$, and its complexity is the number of segments in this path. Lower bounds for trees straightforwardly extend to polygonal instances. Again, we aim to obtain a spanner of complexity close to the lower bound.

▶ **Lemma 14.** *The lower bounds of Lemmata 3, 4, and 5 also hold for simple polygons.*

Spanner construction. Next, we describe how to obtain a low-complexity spanner in a simple polygon using at most k Steiner points. In our approach, we combine ideas from [2] and [15] with the forest spanner of Theorem 13. We first give a short overview of the approach to obtain a low complexity $2\sqrt{2}t$-spanner [15], and then discuss how to combine these ideas with the forest spanner to obtain a low complexity Steiner spanner.

We partition the polygon P into two subpolygons P_ℓ and P_r by a vertical line segment λ such that roughly half of the sites lie in either subpolygon. For the line segment λ, we then consider the following weighted 1-dimensional space. For each site $p \in S$, let p_λ be the *projection* of p: the closest point on λ to p. The (weighted 1-dimensional distance) between two sites p_λ, q_λ is defined as $d_w(p_\lambda, q_\lambda) := d(p, p_\lambda) + d(p_\lambda, q_\lambda) + d(q, q_\lambda)$. In other words, the sites in the 1-dimensional space are weighted by the distance to their original site in P. For this 1-dimensional space we construct a t-spanner $\mathcal{G}_\lambda$, and for each link (p_λ, q_λ) in $\mathcal{G}_\lambda$ we add the link (p, q) to the spanner $\mathcal{G}$. Finally, we process the subpolygons P_ℓ and P_r recursively. De Berg, van Kreveld, and Staals [15] show that this gives a $\sqrt{2}t$-spanner in a simple polygon. To obtain a spanner of complexity $O(mn^{1/t} + n \log^2 n)$, they construct a 1-dimensional $2t$-spanner $\mathcal{G}_\lambda$ using the approach of Lemma 8, resulting in a $2\sqrt{2}t$-spanner.

In our case, we require information on the paths from the sites to their projection instead of only their distance to decide where to place the Steiner points. This information is captured in the shortest path tree SPT_λ of the segment λ, which is the union of all shortest

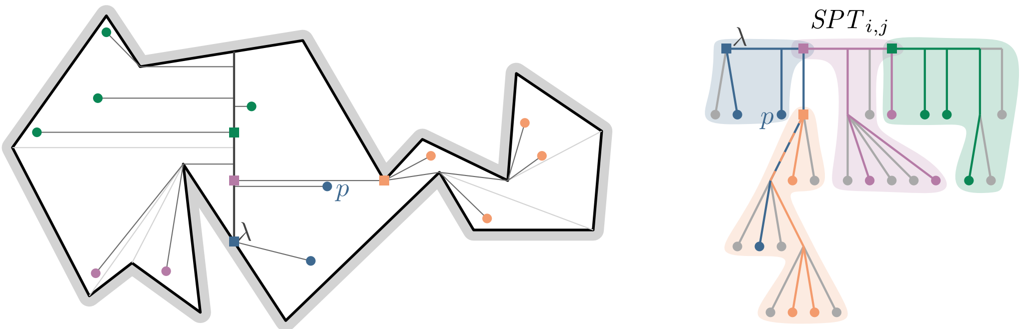


Figure 6 The shortest path tree of λ in P' and its $SPT_{i,j}$. The grey nodes and edges are not included in $SPT_{i,j}$, but can be assigned to a T'_i as indicated by the colored backgrounds. The squares show the Steiner points in $SPT_{i,j}$ and P'. The sites in P' are colored as the trees T'_i.

paths from the vertices of P to their closest point on λ. Additionally, we include all sites in S in the tree SPT_λ. The segment λ is split into multiple edges at the projections of the sites, see Figure 6. The tree SPT_λ is rooted at the lower endpoint of λ and has $O(m+n)$ vertices.

We adapt the algorithm to build a spanner in P as follows. Instead of computing a 1-dimensional spanner directly in each subproblem in the recursion, we first collect the shortest path trees of all subproblems. Let $SPT_{i,j}$ denote the shortest path tree of the j-th subproblem at the i-th level of the recursion. We exclude all vertices from $SPT_{i,j}$ that have no site as a descendant. This ensures that all leaves of the tree are sites. Let $\mathcal{F} = \cup_{i,j} SPT_{i,j}$ be the forest consisting of all trees. A site in S or vertex of P can occur in multiple trees $SPT_{i,j}$, but they are seen as distinct sites and vertices in the forest $\mathcal{F}$. We call a tree $SPT_{i,j}$ *large* if $0 \leq i \leq \log k$ and *small* otherwise. In other words, the trees created in the recursion up to level $\log k$ are large. We then partition $\mathcal{F}$ into two forests $\mathcal{F}_s$ and $\mathcal{F}_\ell$ containing the small and large trees. For each tree in $\mathcal{F}_s$ we directly apply the $2t$-spanner of Lemma 8 that uses no Steiner points to obtain a spanner $\mathcal{G}_s$. For the forest $\mathcal{F}_\ell$ we apply Theorem 13 to obtain a $2t$-spanner $\mathcal{G}_\ell$ for $\mathcal{F}_\ell$. Let $\mathcal{G}_\mathcal{F} = \mathcal{G}_s \cup \mathcal{G}_\ell$. A Steiner point in $\mathcal{G}_\mathcal{F}$ corresponds to either a vertex of P or a point on λ. Let $\mathcal{S}$ denote the set of Steiner points. To obtain a spanner $\mathcal{G}$ in the simple polygon, we add a link (p,q), $p,q \in S \cup \mathcal{S}$, to $\mathcal{G}$ whenever there is a link in $\mathcal{G}_\mathcal{F}$ between (a copy of) p and q.

▶ **Lemma 15.** *The graph $\mathcal{G}$ is a $2\sqrt{2}t$-spanner for the sites S in P of size $O(n \log^2 n)$.*

Complexity analysis. To bound the complexity of the links in $\mathcal{G}$, we have to account for the complexity of links generated by both $\mathcal{G}_s$ and $\mathcal{G}_\ell$. Bounding the complexity of $\mathcal{G}_s$ is relatively straightforward, but to bound the complexity of $\mathcal{G}_\ell$ we first prove a lemma on the structure of a shortest path in P between sites in $\mathcal{F}_\ell$.

Let T be a tree in $\mathcal{F}_\ell$ and let P' be the corresponding subpolygon of the subproblem. We consider the shortest path between two sites that are assigned to the same subtree T'_i of T by the forest algorithm. It can be that this shortest path uses vertices of P' that were excluded from T, as they had no site as a descendant. For the analysis, we do include these vertices in T and assign them to subtrees T'_j as in Section 2.2; see Figure 6. The following lemma states that the complexity of a shortest path between two sites in the same subtree T'_i is bounded by the number of vertices in T'_i. We use this to bound the complexity in Lemma 17.

▶ **Lemma 16.** *A shortest path $\pi(p,q)$ in P' between sites $p,q \in T'_i$ uses vertices in T'_i only.*

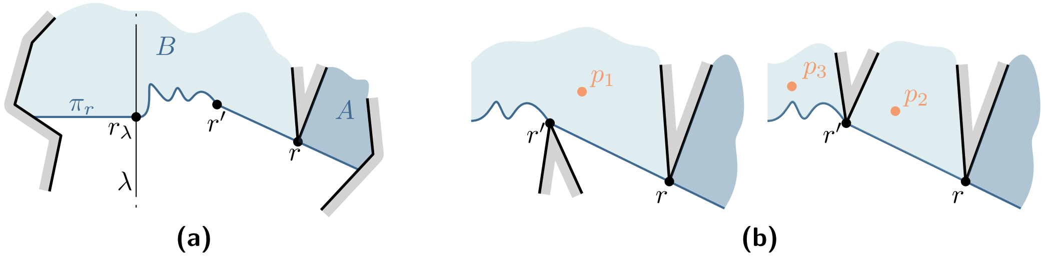


Figure 7 **(a)** The extended path π_r separates the polygon into $P'_r = A \cup B$ and $P'_{\neg r}$. **(b)** Sites $p_{1\ldots3}$ correspond to the respective subcases (i–iii) based on the structure of the polygon around r'.

Proof. Assume for contradiction that r is a highest vertex in T used by $\pi(p,q)$ that is not in T'_i. First, consider the case that r is the root of T. Recall that this means r is the bottom endpoint of λ, and thus lies on the boundary of P'. As r is on $\pi(p,q)$, it must be that p and q lie in different subpolygons, and at least one of them lies below the horizontal line through r. This implies that $s_i = r$, which is a contradiction.

Next, consider the case that r is not the root of T. Let r' be the parent of r. If r' is in T'_i, then it must be a leaf. We consider the following partition of P'. Recall that r_λ denotes the closest point on λ to r. We extend the shortest path $\pi(r, r_\lambda)$ to the boundary of P' by extending the first and last line segments of the path to obtain a path π_r, see Figure 7(a). Let $\partial P'$ denote the boundary of P'. We define P'_r to be the closed polygon bounded by $\partial P'$ and π_r that contains the polygon edges incident to r, and $P'_{\neg r} := P' \setminus P'_r$. Because r is a reflex vertex of P', P'_r is well-defined. Without loss of generality, we assume that P'_r contains the part of λ above r_λ, as in Figure 7(a). If both p and q are in $P'_{\neg r}$, then $r \notin \pi(p,q)$. It follows that p and/or q are in P'_r. Without loss of generality, assume that $p \in P'_r$.

We distinguish two cases based on the location of p, see Figure 7(a). Either $p \in A$, where $A \subset P'_r$ is bounded by the extension segment starting at r and $\partial P'$, or $p \in B$, where $B \subset P'_r$ is bounded by $\pi(r, r_\lambda)$, the extension segment starting at r_λ, and $\partial P'$.

If $p \in A$, then p is a descendant of r in T. As p and q are in T'_i and r is not, it must be that q is also a descendant of r. It follows that $q \in A$, but this means that r is not a reflex vertex on $\pi(p,q)$, which contradicts it being a shortest path.

If $p \in B$, the previous paragraph implies that $q \notin A$. Additionally, $q \notin B$ as well, as r would then not be a reflex vertex in $\pi(p,q)$. It follows that $q \in P'_{\neg r}$. Next, we make a distinction on whether r' is a vertex of P' or not. First, assume that r' is not a vertex of P', and thus $r' \in \lambda$. Because $p \in B$, p_λ must be at or above r'. Because $q \in P'_{\neg r}$, q_λ must be below r'. This implies that the path in T from p to q visits r', which contradicts $p, q \in T'_i$.

Next, we assume that r' is a vertex of P'. We distinguish three different subcases based on the shape of the polygon around r', see Figure 7(b), and find a contradiction in each case:

(i) **The edges of P' incident to r' are in $P'_{\neg r}$.** As r is on $\pi(p,q)$, q must be a descendant of r'. It follows that the Steiner point s_i is located on the path in T from q to r', so p is also a descendant of r'. It follows that p is on the segment rr'. However, for r to be on $\pi(p,q)$, q must then be in A, which is a contradiction.

(ii) **The edges of P' incident to r' are in P'_r, and r' is on $\pi(p, p_\lambda)$.** In this case, p is a descendant of r'. This again implies that q is a descendent of r', which contradicts $q \in P'_{\neg r}$.

(iii) **The edges of P' incident to r' are in P'_r, and r' is not on $\pi(p, p_\lambda)$.** The path $\pi(p,q)$ either intersects the boundary of B twice, which is not allowed as both are shortest paths, or visits r' as well. However, this implies that $q \in A$, which is a contradiction. ◀

▶ **Lemma 17.** *The spanner $\mathcal{G}$ has complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$.*

Proof. To bound the complexity of the links in $\mathcal{G}$ generated by $\mathcal{G}_s$ we apply Lemma 8 directly. As Lemma 8 corresponds to the algorithm to construct a low complexity spanner in a polygon using the shortest path tree, the complexity bound also holds in the simple polygon setting. Using $\sum_{j=0}^{2^i} m_{i,j} = O(m)$, where $m_{i,j}$ is the number of vertices in $SPT_{i,j}$, the complexity is

$$\sum_{i=\log k}^{O(\log n)} \sum_{j=0}^{2^i} O\left(m_{i,j}\left(\frac{n}{2^i}\right)^{1/t} + \frac{n}{2^i}\log\left(\frac{n}{2^i}\right)\right) = O\left(\frac{mn^{1/t}}{k^{1/t}} + n\log^2 n\right).$$

For $\mathcal{F}_\ell$, the algorithm of Lemma 8 is used as a subroutine on every subtree T_i'. Lemma 16 implies that the complexity bound of Theorem 13 also holds for links in P. Recall that the number of sites in $\mathcal{F}_\ell$ is $O(n\log k)$. A vertex of P can occur in at most two subproblems at each level of the recursion that partitions P, thus the number of vertices in $\mathcal{F}_\ell$ is $O((m+n)\log k)$. As the n sites are equally divided over all subproblems at level i, the complexity of the links in $\mathcal{G}$ generated by $\mathcal{G}_\ell$ given by Theorem 13 is improved to

$$O\left(\frac{m\log k(n\log k)^{1/t}}{k^{1/t}} + n\log k\log\left(\frac{n\log k}{k}\right)\right) = O\left(\frac{mn^{1/t}(\log k)^{1+1/t}}{k^{1/t}} + n\log^2 n\right). \blacktriangleleft$$

▶ **Theorem 18.** *Let S be a set of n point sites in a simple polygon P with m vertices, and $t \geq 1$ be any integer constant. For any $1 \leq k \leq n$, we can build a geodesic $2\sqrt{2}t$-spanner with at most k Steiner points, of size $O(n\log^2 n)$ and complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$ in $O(n\log^2 n + m\log n + K)$ time, where K is the output size.*

A relaxed geodesic $(2k+\varepsilon)$-spanner. In a more recent version of the paper by de Berg, van Kreveld, and Staals [14, 15] they show how to apply the refinement proposed by Abam, de Berg, and Seraji [2] to improve the spanning ratio to $(2k+\varepsilon)$ for any constant $\varepsilon \in (0, 2k)$. They make two changes in their approach. First, instead of using the shortest path between two sites as a link they allow a link to be any path between two sites. They call such a spanner a *relaxed* geodesic spanner. Second, for each split of the polygon they construct spanners on several sets of sites in the 1-dimensional weighted space. Using the same adaptations, we obtain a relaxed $(2k+\varepsilon)$-spanner of complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$.

4 Steiner spanners in polygonal domains

If the polygon contains holes, the spanner construction in the previous section no longer suffices. In particular, we may need a different type of separator, and shortest paths in P are no longer restricted to vertices in some subtree (Lemma 16 does not hold). De Berg, van Kreveld, and Staals [15] run into similar problems when generalizing their low complexity spanner, and solve them as follows. There are two main changes in their construction. First, the separator is no longer a line segment, but a *balanced separator* that consists of at most three shortest paths that partition the domain into two subdomains P_r and P_ℓ. They then construct a spanner $\mathcal{G}_\lambda$ on the 1-dimensional space containing the projections of the sites for each shortest path in the separator. Second, the links that are included in the spanner are no longer shortest paths, but consist of at most three shortest paths, resulting in a relaxed geodesic spanner. In contrast to the simple polygon, using a 1-dimensional spanner with spanning ratio t results in a spanning ratio in P of $3t$ [15].

To construct a low complexity spanner using k Steiner points, we use our simple polygon approach with the adaptions of [15]. The number of trees, and thus the number of sites and vertices in the trees, increases by a constant factor, as we create at most three shortest path trees at each level. To bound the complexity, we can no longer apply Lemma 16. However, the links that are added to $\mathcal{G}$ are shortest paths in the shortest path tree. Therefore, the bound on the complexity of $\mathcal{G}_{\mathcal{F}}$ directly translates to a bound on the complexity of $\mathcal{G}$. As in the simple polygon case, we obtain a spanner of complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$.

▶ **Theorem 19.** *Let S be a set of n point sites in a polygonal domain P with m vertices, and $t \geq 1$ be any integer constant. For any $k \leq n$, we can build a relaxed geodesic $6t$-spanner with at most k Steiner points, of size $O(n\log n\log(n/k))$ and complexity $O(mn^{1/t}(\log k)^{1+1/t}/k^{1/t} + n\log^2 n)$ in $O(n\log^2 n + m\log n\log m + K)$ time, where K is the output size.*

5 Future work

On the side of constructing low-complexity spanners, an interesting direction for future work would be to close the gap between the upper and lower bounds, both with and without using Steiner points. We believe it might be possible to increase the $n^{1/(t+1)}$ term to $n^{1/t}$ (or even $n^{1/(t-1)}$) in Lemma 5. On the side of the hardness, many interesting open questions remain, such as: Is the problem still hard in a simple polygon? Can we show hardness for other spanning ratios and/or a less restricted complexity requirement? Is the problem even in NP?

References

1 Mohammad Ali Abam, Marjan Adeli, Hamid Homapour, and Pooya Zafar Asadollahpoor. Geometric spanners for points inside a polygonal domain. In *Proc. 31st International Symposium on Computational Geometry, SoCG*, volume 34 of *LIPIcs*, pages 186–197. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.SOCG.2015.186.

2 Mohammad Ali Abam, Mark de Berg, and Mohammad Javad Rezaei Seraji. Geodesic spanners for points on a polyhedral terrain. *SIAM J. Comput.*, 48(6):1796–1810, 2019. doi:10.1137/18M119358X.

3 Khaled M. Alzoubi, Xiang-Yang Li, Yu Wang, Peng-Jun Wan, and Ophir Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Trans. Parallel Distributed Syst.*, 14(4):408–421, 2003. doi:10.1109/TPDS.2003.1195412.

4 Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel H. M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annual ACM Symposium on Theory of Computing, STOC*, pages 489–498. ACM, 1995. doi:10.1145/225058.225191.

5 Sujoy Bhore and Csaba D. Tóth. Light Euclidean Steiner spanners in the plane. In *Proc. 37th International Symposium on Computational Geometry, SoCG*, volume 189 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.SOCG.2021.15.

6 Glencora Borradaile and David Eppstein. Near-linear-time deterministic plane Steiner spanners for well-spaced point sets. *Comput. Geom.*, 49:8–16, 2015. doi:10.1016/J.COMGEO.2015.04.005.

7 Prosenjit Bose, Joachim Gudmundsson, and Michiel H. M. Smid. Constructing plane spanners of bounded degree and low weight. *Algorithmica*, 42(3-4):249–264, 2005. doi:10.1007/S00453-005-1168-8.

8 Prosenjit Bose and Michiel H. M. Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom.*, 46(7):818–830, 2013. doi:10.1016/J.COMGEO.2013.04.002.

9 T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. *ACM Trans. Algorithms*, 12(4):55:1–55:22, 2016. doi:10.1145/2915183.

10 T.-H. Hubert Chan, Mingfei Li, Li Ning, and Shay Solomon. New doubling spanners: Better and simpler. *SIAM J. Comput.*, 44(1):37–53, 2015. doi:10.1137/130930984.

11 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annual ACM Symposium on Theory of Computing, STOC*, pages 56–65. ACM, 1987.

12 Vincent Cohen-Addad, Arnold Filtser, Philip N. Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *Proc. 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 589–600. IEEE, 2020. doi:10.1109/FOCS46700.2020.00061.

13 Sarita de Berg, Tim Ophelders, Irene Parada, Frank Staals, and Jules Wulms. The complexity of geodesic spanners using steiner points, 2024. arXiv:2402.12110, doi:10.48550/arXiv.2402.12110.

14 Sarita de Berg, Marc van Kreveld, and Frank Staals. The complexity of geodesic spanners. *CoRR*, abs/2303.02997, 2023. URL: https://arxiv.org/abs/2303.02997, doi:10.48550/arXiv.2303.02997.

15 Sarita de Berg, Marc J. van Kreveld, and Frank Staals. The complexity of geodesic spanners. In *Proc. 39th International Symposium on Computational Geometry, SoCG*, volume 258 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.16.

16 Yefim Dinitz, Michael Elkin, and Shay Solomon. Shallow-low-light trees, and tight lower bounds for Euclidean spanners. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 519–528, 2008. doi:10.1109/FOCS.2008.24.

17 Michael Elkin and Shay Solomon. Narrow-shallow-low-light trees with and without Steiner points. *SIAM J. Discret. Math.*, 25(1):181–210, 2011. doi:10.1137/090776147.

18 Michael Elkin and Shay Solomon. Optimal Euclidean spanners: Really short, thin, and lanky. *J. ACM*, 62(5):35:1–35:45, 2015. doi:10.1145/2819008.

19 Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient regression in metric spaces via approximate Lipschitz extension. *IEEE Trans. Inf. Theory*, 63(8):4838–4849, 2017. doi:10.1109/TIT.2017.2713820.

20 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th Annual European Symposium on Algorithms, ESA*, volume 5193 of *LNCS*, pages 478–489, 2008. doi:10.1007/978-3-540-87744-8_40.

21 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

22 Hung Le and Shay Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1078–1100. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00069.

23 Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proc. 13th Annual ACM Symposium on the Theory of Computing, STOC*, pages 186–195. ACM, 1998. doi:10.1145/276698.276734.

24 Joseph S. B. Mitchell and Wolfgang Mulzer. Proximity algorithms. In *Handbook of Discrete and Computational Geometry (3rd Edition)*, chapter 32, pages 849–874. Chapman & Hall/CRC, 2017.

25 Giri Narasimhan and Michiel H. M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

26 Jan Remy, Reto Spöhel, and Andreas Weißl. On Euclidean vehicle routing with allocation. *Comput. Geom.*, 43(4):357–376, 2010. doi:10.1016/J.COMGEO.2008.12.009.

Constrained Boundary Labeling

Thomas Depian
Algorithms and Complexity Group, TU Wien, Austria

Martin Nöllenburg
Algorithms and Complexity Group, TU Wien, Austria

Soeren Terziadis
Algorithms cluster, TU Eindhoven, The Netherlands

Markus Wallinger
Chair for Efficient Algorithms, Technical University of Munich, Germany

Abstract

Boundary labeling is a technique in computational geometry used to label dense sets of feature points in an illustration. It involves placing labels along an axis-aligned bounding box and connecting each label with its corresponding feature point using non-crossing leader lines. Although boundary labeling is well-studied, semantic constraints on the labels have not been investigated thoroughly. In this paper, we introduce *grouping* and *ordering constraints* in boundary labeling: Grouping constraints enforce that all labels in a group are placed consecutively on the boundary, and ordering constraints enforce a partial order over the labels. We show that it is NP-hard to find a labeling for arbitrarily sized labels with unrestricted positions along one side of the boundary. However, we obtain polynomial-time algorithms if we restrict this problem either to uniform-height labels or to a finite set of candidate positions. Finally, we show that finding a labeling on two opposite sides of the boundary is NP-complete, even for uniform-height labels and finite label positions.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry; Theory of computation → Problems, reductions and completeness; Human-centered computing → Geographic visualization

Keywords and phrases Boundary labeling, Grouping constraints, Ordering constraints

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.26

Related Version *Full Version*: `https://arxiv.org/abs/2402.12245` [10]

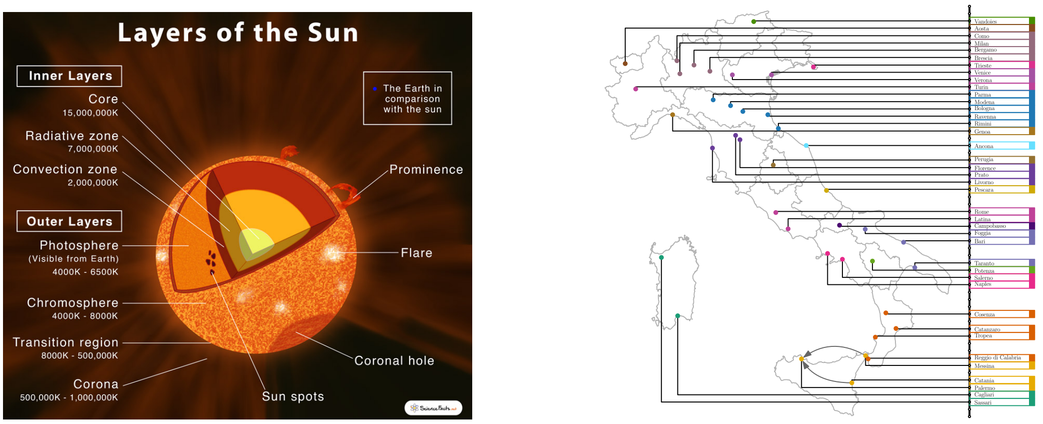
Funding *Thomas Depian*: Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029].
Martin Nöllenburg: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035].
Soeren Terziadis: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035] and European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034253.
Markus Wallinger: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035].

1 Introduction

Annotating features of interest with textual information in illustrations, e.g., in technical, medical, or geographic domains, is an important and challenging task in graphic design and information visualization. One common guideline when creating such labeled illustrations is to "not obscure important details with labels" [9, p. 35]. Therefore, for complex illustrations, designers tend to place the labels outside the illustrations, creating an *external labeling* as shown in Figure 1a. Feature points, called *sites*, are connected to descriptive labels with non-crossing polyline *leaders*, while optimizing an objective function, e.g., the leader length.

External labeling is a well-studied area both from a practical visualization perspective and from a formal algorithmic perspective [5]. One aspect of external labeling that has not yet been thoroughly studied in the literature, though, and which we investigate in this paper,

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 26; pp. 26:1–26:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Schematic of the sun. © ScienceFacts.net [7]; reproduced with permission.

(b) Cities in Italy. Labeling with *po*-leaders created by our algorithm described in Section 2.2.

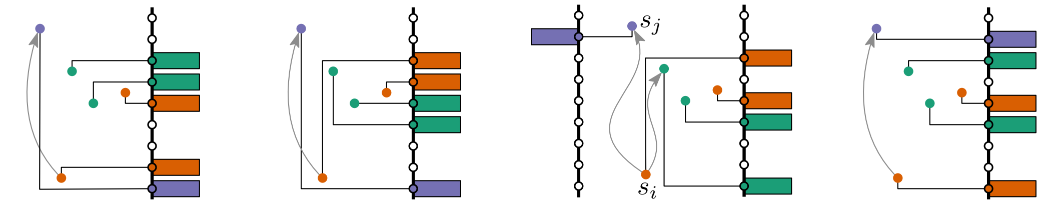
Figure 1 Labelings that adhere to semantic constraints.

is that of constraining the placement of (subsets of) labels in the optimization process. The sequential arrangement of external labels along the boundary of the illustration creates new spatial proximities between the labels that do not necessarily correspond to the geometric proximity patterns of the sites in the illustration. Hence, it is of interest in many applications to put constraints on the grouping and ordering of these labels in order to improve the readability and semantic coherence of the labeled illustration. Examples of such constraints could be to group labels of semantically related sites or to restrict the top-to-bottom order of certain labels to reflect some ordering of their sites in the illustration; see Figure 1a, where the inner and outer layers of the sun are grouped and ordered from the core to the surface.

More precisely, we study such constrained labelings in the *boundary labeling* model, which is a well-studied special case of external labelings. Here, the labels must be placed along a rectangular boundary around the illustration [4]. Initial work placed the labels on one or two sides of the boundary, usually the left and right sides. For uniform-height labels, polynomial-time algorithms to compute a labeling that minimizes the length of the leaders [4] or more general optimization functions [6] have been proposed. Polynomial-time approaches to compute a labeling with equal-sized labels on (up to) all four sides of the boundary are also known [21]. For non-uniform height labels, NP-hardness has been shown in the general two-sided [4], and in different one-sided settings [3, 12]. Several leader styles have been considered, and we refer to the book of Bekos et al. [5] and the user study of Barth et al. [1] for an overview. In this paper, we will focus on a frequently used class of L-shaped leaders, called *po-leaders*, that consist of two segments: one is ***p***arallel and the other ***o***rthogonal to the side of the boundary on which the label is placed [4], see Figure 1b. These *po*-leaders turned out as the recommended leader type in the study of Barth et al. [1] as they performed well in various readability tasks and received high user preference ratings.

The literature considered various extensions of boundary labeling [2, 12, 17, 18], and we broaden this body of work with our paper that aims at systematically investigating the above-mentioned constraints in boundary labeling from an algorithmic perspective.

Problem Description. In the following, we use the taxonomy of Bekos et al. [5] where applicable. Let $\mathcal{S}$ be a set of n sites in $\mathbb{R}^2$ enclosed in an axis-parallel bounding box $\mathcal{B}$ and in general position, i.e., no two sites share the same x- or y-coordinate. For each site $s_i \in \mathcal{S}$, we



(a) Length-minimal. **(b)** Bend-minimal. **(c)** A 2-sided labeling. **(d)** Non-admissible.

Figure 2 Colors indicate grouping and arrows ordering constraints. Labelings that are optimal with respect to **(a)** the total leader length and **(b)** the number of bends. In the 2-sided layout **(c)** the ordering constraint $s_i \preccurlyeq s_j$ is not enforced since ℓ_i and ℓ_j are on different sides. Note that **(d)** is a planar but non-admissible length-minimal labeling.

have an open rectangle ℓ_i of height $h(\ell_i)$ and some width, which we call the *label* of the site. The rectangles model the (textual) labels, which are usually a single line of text in a fixed font size, as their bounding boxes. Hence, we often restrict ourselves to uniform-height labels, but neglect their width. The *po*-leader $\lambda_i = (s_i, c_i)$ is a polyline consisting of (up to) one vertical and one horizontal segment and connects site s_i with the *reference point* c_i, which is a point on the boundary $\mathcal{B}$ at which we attach the label ℓ_i. The point where λ_i touches ℓ_i is called the *port* of ℓ_i. We define the port for each label ℓ to be at half its height, i.e., we use *fixed* ports. Hence, the position of c_i uniquely determines the position of the port p_i and thus the placement of the label ℓ_i. We denote with Λ the set of all possible leaders and with $\mathcal{C}$ the set of all possible reference points. In a *b*-sided boundary labeling $\mathcal{L}\colon \mathcal{S} \to \mathcal{C}$, we route for each site $s \in \mathcal{S}$ a leader λ to the reference point $\mathcal{L}(s)$ on the right ($b = 1$) or the right and left ($b = 2$) side of $\mathcal{B}$, such that we can (in a post-processing step) place the label ℓ for s at the reference point $\mathcal{L}(s)$ and no two labels will overlap. If $\mathcal{C}$ consists of a finite set of m candidate reference points on $\mathcal{B}$, we say that we have *fixed (candidate) reference points*, otherwise $\mathcal{C}$ consists of the respective side(s) of $\mathcal{B}$, which is called *sliding (candidate) reference points*. In the following, we call the reference points in $\mathcal{C}$ simply *candidates*. A labeling is called *planar* if no two labels overlap and there is no leader-leader or leader-site crossing. We can access the x- and y-coordinate of a site, port, or candidate with $x(\cdot)$ and $y(\cdot)$.

In our constrained boundary labeling setting, we are given a tuple of constraints $(\Gamma, \preccurlyeq)$ consisting of a family of k grouping constraints Γ and a partial order $\preccurlyeq$ on the sites. A *grouping* constraint $\emptyset \neq \mathcal{G} \subseteq \mathcal{S}$ enforces that the labels for the sites in $\mathcal{G}$ appear consecutively on the same side of the boundary, as in Figures 2a–2c, but in general there can be gaps between two labels of the same group; compare Figures 2a and 2b. An *ordering* constraint $s_i \preccurlyeq s_j$ enforces for the labeling $\mathcal{L}$ to have $y(\mathcal{L}(s_i)) \geq y(\mathcal{L}(s_j))$ but only if s_i and s_j are labeled on the same side of $\mathcal{B}$. Hence, if the labels ℓ_i and ℓ_j are placed on the same side of $\mathcal{B}$, then the ordering constraint enforces that ℓ_j must not appear above ℓ_i. On the other hand, if ℓ_i and ℓ_j are on different sides of $\mathcal{B}$, then the constraint $s_i \preccurlyeq s_j$ has no effect; see also Figure 2c. This interpretation is motivated by the Gestalt principle of proximity [30], as the spatial distance between labels on opposite sides is usually large and we thus perceive labels on the same side as belonging together. Furthermore, this interpretation of ordering constraints has already been applied in hand-made labelings [14]. Note that in general $\preccurlyeq$ may contain reflexive constraints, which will be fulfilled by any labeling and can thus be removed. The one-sided model in addition implicitly fulfills any transitive constraint. Hence, we work in the one-sided model with the transitive reduction of $(\mathcal{S}, \preccurlyeq)$. In the following, we denote with r the number of ordering constraints in the respective model.

A labeling *respects* the grouping/ordering constraints if all the grouping/ordering constraints are satisfied. Furthermore, the grouping/ordering constraints are *consistent* if there exists a (not necessarily planar) labeling that respects them. Similarly, the constraints $(\Gamma, \preccurlyeq)$ are consistent if there exists a labeling that respects Γ and $\preccurlyeq$ simultaneously. Finally, a labeling is *admissible* if it is planar and respects the constraints. If an admissible labeling exists, we want to optimize a quality criterion expressed by a function $f\colon \Lambda \to \mathbb{R}_0^+$. In this paper, f measures the length or the number of bends of a leader, which are the most commonly used criteria [5]. Figure 2 highlights the differences and shows in (d) that an optimal admissible labeling might be, with respect to f, worse than its planar (non-admissible) counterpart.

In an instance $\mathcal{I}$ of the CONSTRAINED b-SIDED BOUNDARY LABELING problem (b-CBL in short), we want to find an admissible b-sided *po*-labeling $\mathcal{L}^*$ for $\mathcal{I}$ (possibly on a set of m candidates $\mathcal{C}$) that minimizes $\sum_{s \in \mathcal{S}} f((s, \mathcal{L}^*(s))$ or report that no admissible labeling exists.

Related Work on Constrained External Labeling. Our work is in line with (recent) efforts to integrate semantic constraints into the external labeling model. The survey of Bekos et al. [5] reports papers that group labels. Some results consider heuristic label placements in interactive 3D visualizations [19, 20] or group (spatially close) sites together to label them with a single label [11, 24, 28], bundle the leaders [25], or align the labels [29]. These papers are usually targeted at applications and do not aim for exact algorithms or formal complexity bounds. Moreover, the grouping is often not part of the input but rather determined by clustering similar sites. To the best of our knowledge, Niedermann et al. [26] are the first that support the explicit grouping of labels while ensuring a planar labeling. They proposed a contour labeling algorithm, a generalization of boundary labeling, that can be extended to group sets of labels as hard constraints in their model. However, they did not analyze this extension in detail and do not support ordering constraints. Recently, Gedicke et al. [16] tried to maximize the number of respected groups that arise from the spatial proximity of the sites or their semantics, i.e., they reward a labeling also based on the number of consecutive labels from the same group. They disallow assigning a site to more than one group, but see combining spatial and semantic groups as an interesting direction for further research. We work towards that goal, as we allow grouping constraints to overlap. Finally, Klawitter et al. [23] visualized geophylogenies by embedding a binary (phylogenetic) tree on one side of the boundary. Each leaf of the tree corresponds to a site, and we aim to connect them using straight-line leaders with few crossings. These trees implicitly encode grouping constraints, as sites with a short path between their leaves must be labeled close together on the boundary. However, Klawitter et al. not only considered a different optimization function, but restricted themselves to binary trees, which cannot represent all grouping constraints. Overall, we can identify a lack of a systematic investigation of (general) grouping and ordering constraints from an algorithmic perspective in the literature. With this paper, we aim to fill this gap.

Contributions. In Section 2, we take a closer look at 1-CBL. We prove that it is NP-hard to find an admissible labeling with sliding candidates and unrestricted label heights (Section 2.1) and present polynomial-time algorithms for fixed candidates and unrestricted label heights (Section 2.2) and for sliding candidates and uniform-height labels (Section 2.3). To that end, we combine a dynamic program with a novel data structure based on PQ-Trees. As our final contribution, we show in Section 3 that 2-CBL is NP-complete, even for uniform-height labels and fixed candidates. To the best of our knowledge, this is the first two-sided boundary labeling problem that is already NP-hard in such a restricted setting. We summarize our results in Table 1.

Table 1 Our results on b-CBL with n sites, m candidates, and a family Γ of k grouping constraints.

b	candidates	label height	result	reference
1	sliding	non-uniform	NP-hard	Theorem 2.1
1	fixed	non-uniform	$\mathcal{O}(n^5 m^3 \log m + k + \sum_{\mathcal{G}\in\Gamma} \lvert\mathcal{G}\rvert)$	Theorem 2.6
1	sliding	uniform	$\mathcal{O}(n^{11} \log n + k + \sum_{\mathcal{G}_P\in\Gamma} \lvert\mathcal{G}\rvert)$	Theorem 2.9
2	fixed	uniform	NP-complete	Theorem 3.1

Full proofs of statements marked by ★ are deferred to the full version [10].

2 The Constrained One-Sided Boundary Labeling Problem

In Section 2.1, we show that finding an admissible labeling for an instance of 1-CBL is NP-hard. However, by restricting the input to either fixed candidates (Section 2.2) or uniform labels (Section 2.3), we can obtain polynomial-time algorithms.

2.1 Constrained One-Sided Boundary Labeling is NP-hard

To show that it is NP-hard to find an admissible labeling in an instance of 1-CBL, we can reduce from the (weakly) NP-complete problem Partition. Inspired by a reduction by Fink and Suri [12] from Partition to the problem of finding a planar labeling with non-uniform height labels and sliding candidates in the presence of a single obstacle on the plane, we will create a gadget of sites with grouping and ordering constraints that simulates such an obstacle. The following construction is not in general position and contains leader-site crossings. We resolve this issue in the full version [10].

Construction of the Instance. We will first give a reduction that only uses grouping constraints. In the end, we show how we can replace the grouping constraints by ordering constraints. Let $(\mathcal{A} = \{a_1, \dots a_N\}, w \colon \mathcal{A} \to \mathbb{N}^+)$ be an instance of the weakly NP-complete problem Partition, in which we want to know if there exists a subset $\mathcal{A}' \subseteq \mathcal{A}$ such that $\sum_{a\in\mathcal{A}'} w(a) = \sum_{a\in\mathcal{A}\setminus\mathcal{A}'} w(a) = A$, for some integer A, for which we can safely assume $A \geq 6$. We create for each element a_i a site s_i whose corresponding label ℓ_{s_i} has a height of $w(a_i)$, and place the sites on a horizontal line next to each other. Furthermore, we create five sites, b_1 to b_5, with corresponding labels of height 1 for b_1 and b_5, height $\lfloor (A-4)/2 \rfloor$ for b_2 and b_4, and height 2 or 3 for b_3, depending on whether A is even or odd, respectively, and place them as in Figure 3. Observe that the heights of the labels for b_1 to b_5 sum up to A. We create the grouping constraints $\{\{b_1, b_2, b_3\}, \{b_2, b_3, b_4\}, \{b_3, b_4, b_5\}\}$, which enforce that any admissible labeling must label these sites as indicated in Figure 3. Since there is neither an alternative order of the labels nor room to slide around, the labels of b_2 to b_4 must be placed contiguously, without any free space, and at that fixed position on the boundary. Hence, we call the resulting structure a *block*. We create two additional blocks above and below the sites for $\mathcal{A}$ to create two A-high free windows on the boundary; see Figure 3. These windows will be the only place the labels for the sites representing the elements of $\mathcal{A}$ can be. Thus, we can form an equivalence between partitioning the elements of $\mathcal{A}$ into two sets, $\mathcal{A}_1$ and $\mathcal{A}_2$, and placing the labels for the sites in the upper or lower window on the boundary, respectively. Since the windows on the boundary have a height of A, we ensure that the sum of the label heights in each window is exactly A. Finally, note that the grouping

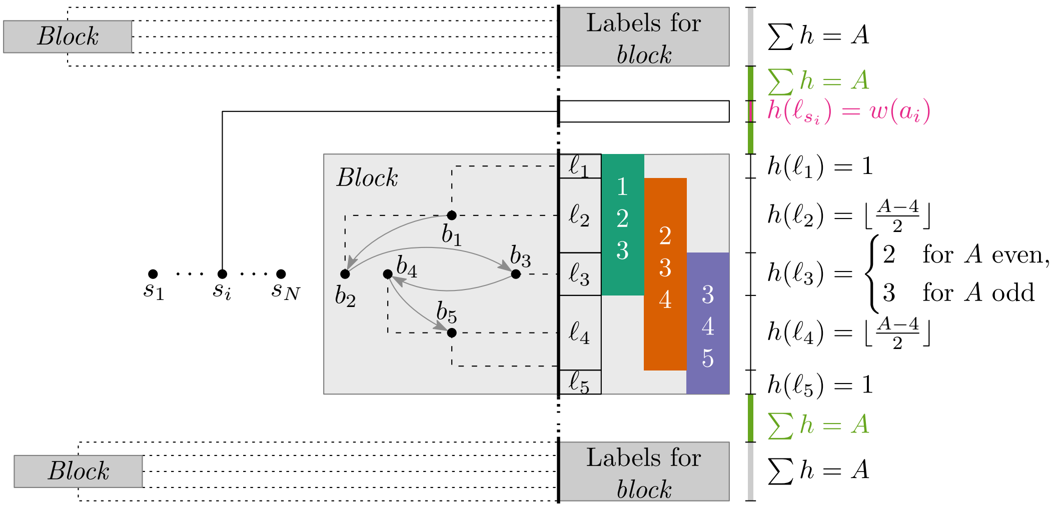


Figure 3 Components of the instance created by our NP-hardness reduction. Colored bands visualize the grouping constraints of a block, that can be replaced by ordering constraints (arrows). Note that the label heights and distances in this figure are not to scale.

constraints we used to keep the labels for the sites of the blocks in place can be exchanged by the ordering constraints $\{b_1 \preccurlyeq b_2, b_2 \preccurlyeq b_3, b_3 \preccurlyeq b_4, b_4 \preccurlyeq b_5\}$ (also shown in Figure 3 via the arrows). Similar substitutions in the other two blocks yield Theorem 2.1.

▶ **Theorem 2.1** (★). *Deciding if an instance of* 1-CBL *has an admissible labeling is* NP*-hard, even for a constant number of grouping or ordering constraints.*

We conclude this section by noting that the problem Partition is only *weakly* NP-complete. Therefore, Theorem 2.1 does not prove that 1-CBL is also NP-hard in the strong sense and such a result would require a reduction from a different problem. Alternatively obtaining a pseudo-polynomial algorithm for our problem at hand would show that it is weakly NP-complete, ruling out strong NP-hardness under common complexity assumptions. We refer to the book by Garey and Johnson [15] for an introduction to NP-completeness and its flavors and leave both directions open for future work.

2.2 Fixed Candidate Reference Points

We assume that we are given a set $\mathcal{C}$ of $m \geq n$ candidates. Benkert et al. [6] observed that in a planar labeling $\mathcal{L}$, the leader λ_L connecting the leftmost site $s_L \in \mathcal{S}$ with some candidate c_L splits the instance $\mathcal{I}$ into two independent sub-instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, excluding s_L and c_L. Therefore, we can describe a sub-instance I of $\mathcal{I}$ by two leaders (s_1, c_1) and (s_2, c_2) that bound the sub-instance from above and below, respectively. We denote the sub-instance as $I = (s_1, c_1, s_2, c_2)$ and refer with $\mathcal{S}(I)$ and $\mathcal{C}(I)$ to the sites and candidates in I, *excluding* those used in the definition of I, i.e., $\mathcal{S}(I) := \{s \in \mathcal{S} \mid x(s_1) < x(s),\ x(s_2) < x(s),\ y(c_2) < y(s) < y(c_1)\}$ and $\mathcal{C}(I) := \{c \in \mathcal{C} \mid y(c_2) < y(c) < y(c_1)\}$. Similarly, for a leader $\lambda = (s, c)$, we say that a site s' with $x(s) < x(s')$ is *above* λ if $y(s') > y(c)$ holds and *below* λ if $y(s') < y(c)$ holds. See also Figure 4 for an illustration of these definitions and notions.

Two more observations about admissible labelings can be made: First, λ_L never splits sites $s, s' \in \mathcal{G}$ with $s_L \notin \mathcal{G}$ for a $\mathcal{G} \in \Gamma$. Second, λ_L never splits sites $s, s' \in \mathcal{S}$ with s above λ_L and s' below λ_L, for which we have $s' \preccurlyeq s_L$, $s' \preccurlyeq s$, or $s_L \preccurlyeq s$. Now, we could immediately

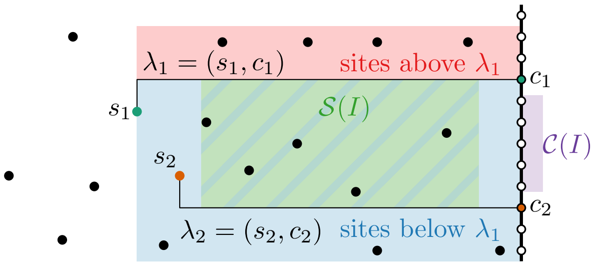


Figure 4 A sub-instance $I = (s_1, c_1, s_2, c_2)$ of our DP-algorithm and the used notation.

define a dynamic programming (DP) algorithm that evaluates the induced sub-instances for each leader that adheres to these observations. However, we would then check every constraint in each sub-instance and not make use of implicit constraints given by, for example, overlapping groups. The following data structure makes these implicit constraints explicit.

PQ-A-Graphs. Every labeling $\mathcal{L}$ induces a permutation π of the sites by reading the labels from top to bottom. Assume that we have at least one grouping constraint, i.e., $k > 0$, and let $M(\mathcal{S}, \Gamma)$ be an $n \times k$ binary matrix with $m_{i,j} = 1$ if and only if $s_i \in \mathcal{G}_j$ for $\mathcal{G}_j \in \Gamma$. We call $M(\mathcal{S}, \Gamma)$ the *sites vs. groups* matrix, and observe that $\mathcal{L}$ satisfies the constraint $\mathcal{G}_j$ if and only if the ones in the column j of $M(\mathcal{S}, \Gamma)$ are consecutive after we order the rows of $M(\mathcal{S}, \Gamma)$ according to π. If a permutation π exists such that this holds for all columns of $M(\mathcal{S}, \Gamma)$, then the matrix has the so-called *consecutive ones property* (*C1P*) [13]. This brings us to the following observation.

▶ **Observation 2.2.** Γ *are consistent for* $\mathcal{S}$ *if and only if* $M(\mathcal{S}, \Gamma)$ *has the C1P.*

Booth and Lueker [8] proposed an algorithm to check whether a binary matrix has the C1P. They use a PQ-Tree to keep track of the allowed row permutations. A *PQ-Tree* τ, for a given set $\mathcal{A}$ of elements, is a rooted tree with one leaf for each element of $\mathcal{A}$ and two different types of internal nodes t: P-nodes, that allow to freely permute the children of t, and Q-nodes, where the children of t can only be inversed [8]. Observation 2.2 tells us that each family of consistent grouping constraints can be represented by a PQ-Tree. Note that we can interpret each subtree of the PQ-Tree as a grouping constraint and we call the resulting grouping constraints *canonical groups*. However, while every canonical group is a grouping constraint, not every given grouping constraint manifests in a canonical group.

While Observation 2.2 implies that PQ-Trees can represent families of consistent grouping constraints, it is folklore that directed acyclic graphs can be used to represent partial orders, i.e., our ordering constraints. We now combine these two data structures into *PQ-A-Graphs*.

▶ **Definition 2.3** (PQ-A-Graph). *Let* $\mathcal{S}$ *be a set of sites,* Γ *be a family of consistent grouping constraints, and* $\preccurlyeq$ *be a partial order on* $\mathcal{S}$*. The* PQ-A-Graph $\mathcal{T} = (\tau, A)$ *consists of the PQ-Tree* τ *for* Γ*, on whose leaves we embed the arcs* A *of a directed graph representing* $\preccurlyeq$*.*

We denote with T_i the *subtree* in the underlying PQ-Tree τ rooted at the node t_i and with leaves(T_i) the leaf set of T_i. Figure 5 shows a PQ-A-Graph and the introduced terminology. Furthermore, observe that checking on the consistency of $(\Gamma, \preccurlyeq)$ is equivalent to solving the Reorder problem on τ and $\preccurlyeq$, i.e., asking whether we can re-order leaves(τ) such that the order induced by reading them from left to right extends the partial order $\preccurlyeq$ [22].

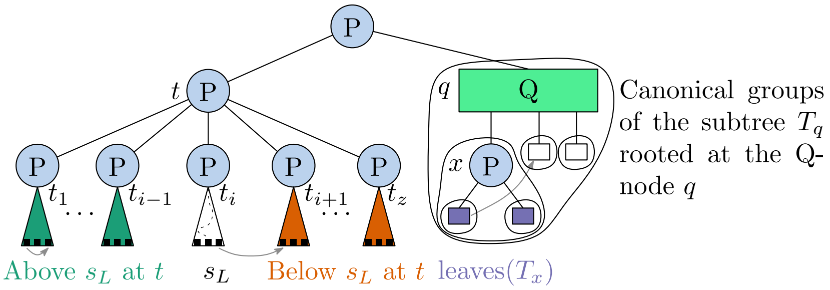


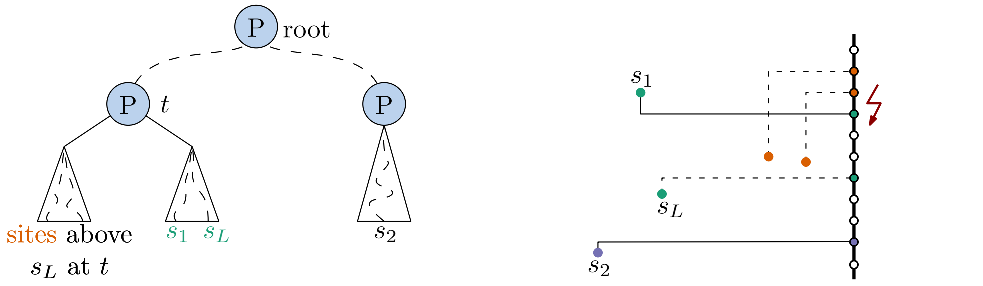
Figure 5 A sample PQ-A-Graph together with the used terminology. Leaves are indicated by squares and ordering constraints by the arrows.

▶ **Lemma 2.4** (★). *Let $\mathcal{S}$ be a set of n sites, Γ be k grouping constraints and $\preccurlyeq$ be r ordering constraints. We can check whether $(\Gamma, \preccurlyeq)$ are consistent for $\mathcal{S}$ and, if so, create the PQ-A-Graph $\mathcal{T}$ in $\mathcal{O}(n + k + r + \sum_{\mathcal{G}\in\Gamma} |\mathcal{G}|)$ time. $\mathcal{T}$ uses $\mathcal{O}(n + r)$ space.*

The Dynamic Programming Algorithm. Let $I = (s_1, c_1, s_2, c_2)$ be a sub-instance and s_L the leftmost site in $\mathcal{S}(I)$. Furthermore, let $\mathcal{T}(s_1, s_2)$ denote the subgraph of the PQ-A-Graph $\mathcal{T}$ rooted at the lowest common ancestor of s_1 and s_2 in $\mathcal{T}$. Note that $\mathcal{T}(s_1, s_2)$ contains at least the sites in $\mathcal{S}(I)$, together with s_1 and s_2. Therefore, it contains all constraints relevant for the sub-instance I. Other constraints either do not affect sites in I, i.e., are represented by other parts of $\mathcal{T}$, or are trivially satisfied. In particular, observe that all constraints induced by nodes further up in $\mathcal{T}$ affect a super-set of $\mathcal{S}(I)$ and in particular have been checked in some sub-instance I' that contains I, i.e., that contains s_1, s_2, c_1, and c_2. Now imagine that we want to place the label ℓ_L for s_L at the candidate $c_L \in \mathcal{C}(I)$. We have to ensure that $\lambda_L = (s_L, c_L)$ does not violate planarity with respect to the already fixed labeling and that in the resulting sub-instances there are enough candidates for the respective sites. Let $\textsc{Admissible}(I, \mathcal{T}, c_L)$ be a procedure that checks this and, in addition, verifies that c_L respects the constraints expressed by $\mathcal{T}(s_1, s_2)$. To do the latter efficiently, we make use of the procedure $\textsc{RespectsConstraints}(I, \mathcal{T}, \lambda_L)$ defined as follows.

Let t_L be the leaf for s_L in $\mathcal{T}(s_1, s_2)$. There is a unique path from t_L to root$(\mathcal{T}(s_1, s_2))$, which we traverse bottom up and consider each internal node on it. Let t be such a node with the children $t_1, \ldots, t_z$ in this order from left to right. Let T_i, $1 \le i \le z$, be the subtree that contains the site s_L, rooted at t_i. The labels for all sites represented by leaves$(T_1), \ldots,$ leaves(T_{i-1}) will be placed above ℓ_L in any labeling $\mathcal{L}$ of $\mathcal{S}$ in which the children of t are ordered as stated. Therefore, we call these sites *above* s_L (at t). Analogously, the sites represented by leaves$(T_{i+1}), \ldots,$ leaves(T_z) are *below* s_L (at t). Figure 5 illustrates this. The sites represented by leaves(T_i) are neither above nor below s_L at t. It is important to note that we use two different notions of *above*/*below*. On the one hand, sites can be above a node t in the PQ-A-Graph $\mathcal{T}(s_1, s_2)$, which depends on the order of the children of t. On the other hand, a site can also be above a leader λ, which depends on the (geometric) position of λ and is independent of $\mathcal{T}(s_1, s_2)$. Recall Figure 5 and compare it with Figure 4 for the former and latter notion of above and below, respectively.

If t is a P-node, we seek a permutation π of the children $t_1, \ldots, t_z$ of t in which *all* the sites in $\mathcal{S}(I)$ above s_L at t (in the permutation π) are above λ_L, and all the sites in $\mathcal{S}(I)$ below s_L at t (in the permutation π) are below λ_L. This means that it cannot be the case that some sites of the same subtree T that does not contain s_L are above λ_L, while others are below λ_L, as this implies that we violate the canonical grouping constraint induced by T.



(a) The PQ-A-Graph $\mathcal{T}(s_1, s_2)$, rooted at root$(\mathcal{T}(s_1, s_2))$, with $s_1 \in$ leaves(T_i).

(b) A (sub-)instance I, where a wrong permutation of the children of t in $\mathcal{T}(s_1, s_2)$ from **(a)** would label the orange sites outside I.

Figure 6 In this situation, C_{above} must not contain subtrees with sites from the sub-instance.

To not iterate through all possible permutations, we distribute the children of t, except t_i, into two sets, C_{above} and C_{below}, depending on whether the sites they represent should be above or below s_L at t. Unless specified otherwise, whenever we mention in the following the site s_1, then the same applies to s_2, but possibly after exchanging *above* with *below*.

If T_j, rooted at a child t_j of t, $1 \leq j \leq z$, $i \neq j$, only contains sites from $\mathcal{S}(I)$, we check whether all the sites are above λ_L, or if all are below λ_L. In the former case, we put t_j in the set C_{above}, and in the latter case in C_{below}. If neither of these cases applies, we return with failure as λ_L splits a (canonical) group to which s_L does not belong. If T_j contains only sites outside the sub-instance and not s_1, we immediately put it in C_{above}. However, if T_j contains $s_1 \in$ leaves(T_j), it can contain some sites in I and others outside I. As s_1 is above s_L at t by the definition of I, we must put t_j in C_{above}. Hence, we check whether all sites in leaves$(T_j) \cap \mathcal{S}(I)$ are above λ_L, i.e., whether they are indeed above s_L at t. Furthermore, if $s_1 \in$ leaves(T_i) holds, then C_{above} must not contain a child t_j containing sites from $\mathcal{S}(I)$, as they would then be labeled outside I, violating the definition of I; see Figure 6. Once t is the root of $\mathcal{T}(s_1, s_2)$, sites from T_j but outside I can be above or below s_L at t, as s_1 *and* s_2 are in the subtree rooted at t. If T_j does not contain the sites s_1 and s_2, and only sites outside I, then the leaders from s_1 or s_2 separate the sites from T_j and $\mathcal{S}(I)$. As these sites together were part of a bigger instance I' (that contains I), for which we already ensured that potential constraints relating a site from T_j and one from $\mathcal{S}(I)$ are respected, we can ignore t_j. In any other case, we perform as described above.

The checks that have to be performed if t is a Q-node are conceptually the same, but simpler, since Q-nodes only allow to inverse the order: Either all sites above s_L at t are above λ_L, and all sites below s_L at t are below λ_L, or all sites above s_L at t are below λ_L, and all sites below s_L at t are above λ_L. In the former case, we keep the order of the children at the node t as they are. In the latter case, we inverse the order of the children at the node t. Note that if a child of t contains the sites s_1, s_2, or sites outside the sub-instance I but in $\mathcal{T}(s_1, s_2)$, one of the two allowed inversions is enforced by the definition of I.

Until now we only verified that we adhere to the grouping constraints. To ensure that we do not violate an ordering constraint, we maintain a look-up table that stores for each site whether it belongs to C_{above}, C_{below}, or T_i. Then, we check for each of the ordering constraints in $\mathcal{T}(s_1, s_2)$ in constant time whether we violate it or not. Note that we violate an ordering constraint if the corresponding arc runs from C_{below} to T_i or to C_{above}, or from T_i to C_{above}. We observe that we query the position of each site s $\mathcal{O}(1)$ times and determine each time its position with respect to the leader λ_L or check whether it is in the sub-instance.

Afterwards, we do not consider this site anymore. As each ordering constraint can be checked in $\mathcal{O}(1)$ time, we have an overall running time of $\mathcal{O}(n + r)$ for the checks at a node t of $\mathcal{T}(s_1, s_2)$, which already includes the computation of the look-up tables.

We say that c_L *respects the constraints for* s_L imposed by $\mathcal{T}(s_1, s_2)$ in the sub-instance $I = (s_1, c_1, s_2, c_2)$ if it respects them at every node t on the path from s_L to the root of $\mathcal{T}(s_1, s_2)$. $\textsc{RespectsConstraints}(I, \mathcal{T}, \lambda_L)$ performs these checks for each node on the path from s_L to $\mathrm{root}(\mathcal{T}(s_1, s_2))$. The length of this path is bounded by the depth of $\mathcal{T}(s_1, s_2)$ which is in $\mathcal{O}(n)$. Hence, $\textsc{RespectsConstraints}(I, \mathcal{T}, \lambda_L)$ runs in $\mathcal{O}(n\,(n + r))$ time. In the following lemma, we show that $\textsc{Admissible}(I, \mathcal{T}, c_L)$ takes $\mathcal{O}(n^2 + nr + \log m)$ time.

▶ **Lemma 2.5 (★).** *Let $I = (s_1, c_1, s_2, c_2)$ be a sub-instance of our DP-Algorithm with the constraints expressed by a PQ-A-Graph $\mathcal{T}$. We can check whether the candidate $c_L \in \mathcal{C}(I)$ is admissible for the leftmost site $s_L \in \mathcal{S}(I)$ using* $\textsc{Admissible}(I, \mathcal{T}, c_L)$ *in $\mathcal{O}(n^2 + nr + \log m)$ time, where $n = |\mathcal{S}|$, $m = |\mathcal{C}|$, and r is the number of ordering constraints.*

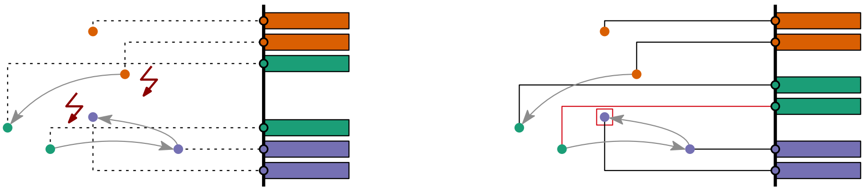
For a sub-instance $I = (s_1, c_1, s_2, c_2)$, we store in a table D the value $f(\mathcal{L}^*)$ of an optimal admissible labeling $\mathcal{L}^*$ on I or ∞ if none exists. If I does not contain a site we set $D[I] = 0$. Otherwise, we use the following relation, where the minimum of the empty set is ∞.

$$D[I] = \min_{\substack{c_L \in \mathcal{C}(I) \text{ where} \\ \textsc{Admissible}(I, \mathcal{T}, c_L) \text{ is true}}} \left(D[(s_1, c_1, s_L, c_L)] + D[(s_L, c_L, s_2, c_2)]\right) + f((s_L, c_L))$$

To show correctness of our DP-Algorithm, one can use a proof analogous to the one of Benkert et al. [6], who propose a similar dynamic program to compute a one-sided labeling with *po*-leaders and a similar-structured optimization function, combined with the fact that we consider only those candidates that are admissible for s_L. By adding artificial sites s_0 and s_{n+1}, and candidates c_0 and c_{m+1}, that bound the instance from above and below, we can describe any sub-instance by a tuple $I = (s_1, c_1, s_2, c_2)$, and in particular the sub-instance for $\mathcal{I}$ by $I_0 = (s_0, c_0, s_{n+1}, c_{m+1})$. As two sites and two candidates describe a sub-instance, there are up to $\mathcal{O}(n^2 m^2)$ possible sub-instances to evaluate. We then fill the table D top-down using memoization. This guarantees us that we have to evaluate each sub-instance I at most once, and only those that arise from admissible candidates. The running time of evaluating a single sub-instance is dominated by the time required to determine for each candidate whether it is admissible. Combined with the size of the table D, we get the following.

▶ **Theorem 2.6 (★).** 1-CBL *for n sites, m* fixed candidates, *r ordering, and k grouping constraints Γ can be solved in $\mathcal{O}(n^5 m^3 \log m + k + \sum_{\mathcal{G} \in \Gamma} |\mathcal{G}|)$ time and $\mathcal{O}(n^2 m^2)$ space.*

Real-world instances often consist of less than 50 sites [26] and we do not expect the number of candidates to be significantly larger than the number of sites. Hence, the running time of Theorem 2.6 does not immediately rule out the practical applicability of our results. Indeed, initial experiments for uniform-height labels confirmed that our algorithm terminates within a few seconds for instances with realistic sizes [1, 16] of up to 25 sites and 50 candidates; see the full version [10] for details and Figure 1b for an example. In fact, dynamic programming is frequently used to obtain exact polynomial-time algorithms in external labeling [5] and it is not uncommon that such algorithms have high running times of up to $\mathcal{O}(n^6)$ and $\mathcal{O}(n^9)$ for the one- and two-sided setting with *po*-leaders, respectively [6, 12, 21, 23]. Finally, we observed in our experiments that the position of the candidates influenced the feasibility of an instance, which makes considering sliding candidates interesting and relevant.



(a) We cannot avoid crossings if we want to respect the constraints.

(b) An alternative set of candidates to **(a)** which allows for an admissible labeling.

Figure 7 An instance whose admissibility depends on the position of the candidates.

2.3 Sliding Candidate Reference Points with Uniform-Height Labels

Fixed candidates have the limitation that the admissibility of an instance depends on the choice and position of the candidates, as Figure 7 shows. By allowing the labels to slide along a sufficiently long vertical boundary line, we remove this limitation. To avoid the NP-hardness shown in Section 2.1, we require that all labels now have uniform height $h > 0$.

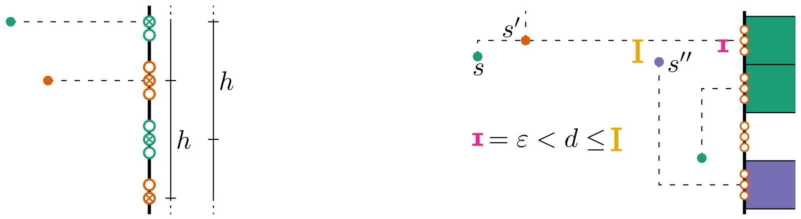
In this section, we will define for each site s a set of $\mathcal{O}(n)$ candidates placed at multiples of h away from $y(s)$, building on an idea of Fink and Suri [12] shown in Figure 8a with the crossed candidates. After extending them by some offset $\varepsilon > 0$ we will prove in order: That if an instance has an admissible labeling, it also has an admissible (bend-minimal) labeling using these candidates (Lemma 2.7), that if such an instance has an admissible labeling in which every leader has a minimum distance to every non-incident site, then there is a labeling with the same property using a slightly different set of candidates, which also has equal or smaller total leader length (Lemma 2.8) and that these results in concert with Theorem 2.6 can be used to solve 1-CBL with uniform-height labels in polynomial time (Theorem 2.9).

Let d be defined as $d := \min_{s,s' \in \mathcal{S}} (|y(s) - y(s')| - qh)$, where $q = \lfloor |y(s) - y(s')| / h \rfloor$, i.e., the smallest distance one must move some site (down) in the instance such that it is vertically a multiple of h away from some other site. For the following arguments to work, we require $d > 0$. However, this can easily be ensured by enforcing that the vertical distance $|y(s) - y(s')|$ between any pair of sites s and s' is not a multiple of h, which can be achieved by perturbating some sites slightly. As we then have $d > 0$, we can select an ε with $0 < \varepsilon < d$. We now define a set of $\mathcal{O}(n^2)$ candidates such that there exists an admissible labeling on these (fixed) candidates, if the instance with sliding candidates possesses an admissible labeling. For each $s \in \mathcal{S}$, we define the set $\mathcal{C}(s) := \{y(s){+}ih,\ y(s){+}ih{\pm}\varepsilon,\ y(s){-}ih,\ y(s){-}ih{\pm}\varepsilon \mid 0 \leq i \leq n\}$ of candidates. Now, we define the set of *canonical candidates* $\mathcal{C}(\mathcal{S})$ as $\mathcal{C}(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{C}(s)$, some of which are depicted in Figure 8a, and show the following.

▶ **Lemma 2.7** (★). *Let $\mathcal{I}$ be an instance of* 1-CBL *with uniform-height labels and sliding candidates. If $\mathcal{I}$ possesses an admissible labeling, it also has one with candidates from $\mathcal{C}(\mathcal{S})$.*

Proof Sketch. Our proof builds on arguments used by Fink and Suri for a similar result [12] and we call a maximal set of touching (but non-overlapping) labels a *stack* [27]. The idea is to transform an admissible labeling $\mathcal{L}$ into an admissible labeling $\mathcal{L}'$ in which each candidate is from $\mathcal{C}(\mathcal{S})$. Labels at a candidate above the candidate $c \in \mathcal{C}(\mathcal{S})$ with $y(c) = y(s_t) + h$ for the top-most site s_t are arranged in $\mathcal{L}'$ as a single stack so that the bottom-most candidate coincides with c. We arrange labels at a candidate at least h below the bottom-most site symmetrically. As they are located above and below all sites, this cannot introduce crossings.

For the remaining labels in $\mathcal{L}$, we iteratively take the bottom-most not yet moved label ℓ and move it upwards until we either hit a candidate from $\mathcal{C}(\mathcal{S})$ or another label ℓ'. In the latter case, we "merge" ℓ and ℓ' into a stack and move them from now on simultaneously. We



(a) Induced candidates as in [12] (marked by a cross) and the extension to canonical candidates.

(b) The leader of s crosses s' after moving labels upwards. We show the candidates from $\mathcal{C}(s')$.

Figure 8 The set of reference points we construct and **(b)** their usage in the proofs.

continue moving the remaining labels in the same manner. Observe that we never move a label past a site, but might introduce leader-site crossings. They can arise if in $\mathcal{L}$ a leader λ from a site s passes between a candidate of $\mathcal{C}(\mathcal{S})$ and a site $s' \neq s$. There, the first candidate that we hit for ℓ might be induced by s' and λ could now cross s'. In such a case, we take that label and its potential stack and move it downwards until we hit a candidate from $\mathcal{C}(\mathcal{S})$. By our selection of ε, it is guaranteed that we will hit a candidate before any other site, since there is at least one other candidate strictly between the end of the stack and any other site s''. We already indicated this in Figure 8a, but show it in more detail in Figure 8b with the orange candidates. After moving all such labels simultaneously, we obtain the labeling $\mathcal{L}'$ where each label is at a candidate from $\mathcal{C}(\mathcal{S})$. Since the relative placement of the labels in $\mathcal{L}'$ is identical to $\mathcal{L}$, all constraints are still respected and $\mathcal{L}'$ is admissible. ◀

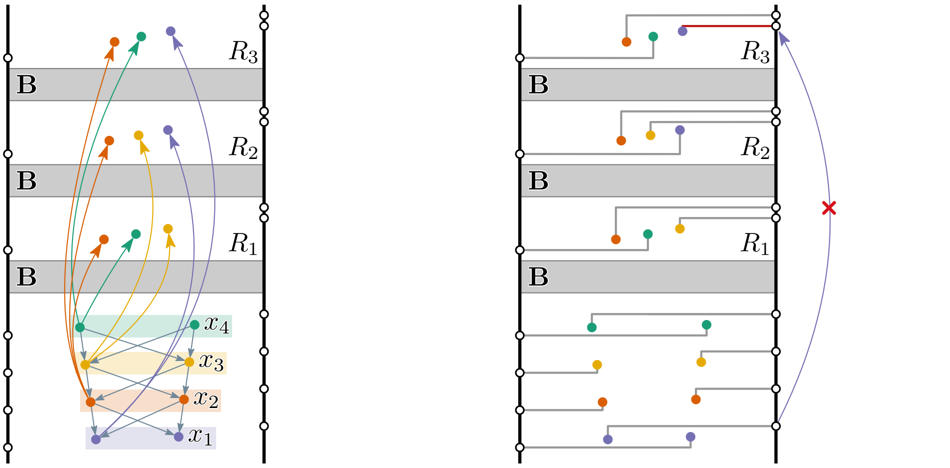
Observe that if we want to obtain a labeling with the minimum number of bends, then it only matters whether a site s is labeled at a candidate c with $y(s) = y(c)$. Labeling s at any other candidate $c' \neq c$ contributes with one bend to the labeling. As each site $s \in \mathcal{S}$ induces a candidate $c \in \mathcal{C}(\mathcal{S})$ with $y(s) = y(c)$, our set of canonical candidates allows for a bend-minimal labeling. For length-minimal labelings, instances without optimal labelings exist. See for example Figure 7b, where we can always move the red leader by a small $\varepsilon' > 0$ closer to the purple site marked with a red box. To ensure the existence of length-minimal labelings, we enforce that the leaders maintain a minimum vertical distance of $v_{\min} > 0$ to non-incident sites. We define $\mathcal{C}'(s) := \{y(s)+qh,\ y(s)+qh \pm v_{\min},\ y(s)-qh,\ y(s)-qh \pm v_{\min} \mid 0 \leq q \leq n\}$, which is an alternative set of canonical candidates that takes $v_{\min}$ into account. Equipped with $\mathcal{C}'(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{C}'(s)$, we can show Lemma 2.8, which is a variant of Lemma 2.7.

▶ **Lemma 2.8 (★).** *Let $\mathcal{I}$ be an instance of* 1-CBL *with uniform-height labels and where the leaders must maintain a vertical distance of at least $v_{\min}$ to non-incident sites. If $\mathcal{I}$ possesses an admissible labeling $\mathcal{L}$ with sliding candidates, then we can find an admissible labeling $\mathcal{L}'$ with candidates from $\mathcal{C}'(\mathcal{S})$ such that the leader length of $\mathcal{L}'$ is at most the one of $\mathcal{L}$.*

Note that $\mathcal{C}'(\mathcal{S})$ can contain candidates for which leaders would not satisfy the requirement on a vertical distance of at least $v_{\min}$ to non-incident sites. Recall that we never moved past a candidate while sliding labels and created in $\mathcal{C}'(\mathcal{S})$ candidates that are $v_{\min}$ away from sites. Hence, we ensure that $\mathcal{L}'$ satisfies this additional criterion as well. This criterion can also be patched into the DP-algorithm without affecting its running time. Finally, this additional criterion is not a limitation, as this is already often required in real-world labelings [26].

With Lemmas 2.7 and 2.8 at hand, we can show the following theorem.

▶ **Theorem 2.9.** 1-CBL *for n sites with* uniform-height labels*, k grouping, and r ordering constraints can be solved in $\mathcal{O}(n^{11} \log n + k + \sum_{\mathcal{G} \in \Gamma} |\mathcal{G}|)$ time and $\mathcal{O}(n^6)$ space.*



(a) The constructed instance.

(b) A (non-admissible) labeling that violates an ordering constraint.

Figure 9 The instance created by our reduction for the formula $R_1 \wedge R_2 \wedge R_3 = (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$. The variable gadgets for x_1, x_2, x_3, and x_4 as well as their occurrences in clause gadgets are drawn in blue, red, yellow, and green, respectively. Ordering constraints between variables and their occurrences in clauses are indicated in (a). If two variables of a clause are set to true, we violate at least one ordering constraint as highlighted with the red leader for the purple site in the clause R_3 in (b). The labeling from (b) induces the variable assignment $x_1 = x_4 =$ true and $x_2 = x_3 =$ false, which does not satisfy $R_3 = (x_1 \vee x_2 \vee x_4)$ as x_1 and x_4 are set to true.

Proof. We note that $\mathcal{C}(\mathcal{S})$ and $\mathcal{C}'(\mathcal{S})$ consist of $\mathcal{O}(n^2)$ canonical candidates. Lemmas 2.7 and 2.8 ensure that we can obtain on these candidates an admissible, bend-, and length-minimal labeling, if there exists one at all. Thus, we can use our DP-Algorithm from Section 2.2 and obtain Theorem 2.9 by plugging in $m = \mathcal{O}(n^2)$ in Theorem 2.6. ◀

3 Constrained Two-Sided Boundary Labeling is NP-complete

In the previous section, we showed that 1-CBL, while generally NP-hard, can be solved efficiently if we have either fixed candidates or labels of uniform height. This does not extend to the generalization of the problem to two sides, as the following theorem underlines.

▶ **Theorem 3.1** (★). *Deciding if an instance of* 2-CBL *has an admissible labeling is* NP*-complete, even for uniform-height labels and fixed candidates.*

Proof Sketch. NP-membership follows from the definition of admissibility, and NP-hardness can be shown by reducing from Positive 1-In-3 Sat, see Figure 9 for an example. The main building block is the *blocker gadget* B, which consists of eight sites and uses grouping and ordering constraints to let the orthogonal parts of its leaders span the entire width between the two boundaries. This enforces that sites on one side of the gadget cannot be labeled on the other side and vice versa. We use this to divide the space between the vertical boundaries into separate strips: one per clause R_i and one at the bottom for all variables. The former contains three *clause sites*, one per occurring variable, of which only one can be labeled on the left side. The latter contains per variable two *variable sites* which encode if the variable is true (a site is labeled on the right side) or false (it is labeled on the left side). Ordering constrains force any variable site to be labeled above any corresponding clause site. The blockers make this impossible, thus forcing all clause sites to be labeled on the opposite side of their variable site, creating a correspondence to a consistent variable assignment. Finally, as in every clause strip, there is only one candidate on the left side, exactly one clause site can be labeled on the left, corresponding to the (single) variable satisfying this clause. ◀

4 Conclusion

We introduced and studied grouping and ordering constraints in boundary labeling. While finding an admissible labeling is NP-hard in general, polynomial-time algorithms for one-sided instances with fixed candidates or uniform-height labels exist. Future work could try to speed up the admissibility checks in our dynamic program to reduce its overall running time or investigate the incorporation of *soft* constraints, i.e., consider the task of maximizing the number of satisfied constraints. Since we can also label features other than points, it is worth studying a variant of this problem with uncertain or variable site locations. Similarly, the support of other leader types or entire other external labeling styles should be investigated.

References

1 Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. On the readability of leaders in boundary labeling. *Information Visualization*, 18(1):110–132, 2019. doi:10.1177/1473871618799500.

2 Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. Many-to-One Boundary Labeling with Backbones. *Journal of Graph Algorithms and Applications (JGAA)*, 19(3):779–816, 2015. doi:10.7155/jgaa.00379.

3 Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary Labeling with Octilinear Leaders. *Algorithmica*, 57(3):436–461, 2010. doi:10.1007/s00453-009-9283-6.

4 Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215–236, 2007. doi:10.1016/j.comgeo.2006.05.003.

5 Michael A. Bekos, Benjamin Niedermann, and Martin Nöllenburg. *External Labeling: Fundamental Concepts and Algorithmic Techniques*. Synthesis Lectures on Visualization. Springer, 2021. doi:10.1007/978-3-031-02609-6.

6 Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for Multi-Criteria Boundary Labeling. *Journal of Graph Algorithms and Applications (JGAA)*, 13(3):289–317, 2009. doi:10.7155/jgaa.00189.

7 Satyam Bhuyan and Santanu Mukherjee (sciencefacts.net). Layers of the Sun, 2023. Accessed on 2023-09-07. URL: https://www.sciencefacts.net/layers-of-the-sun.html.

8 Kellogg S. Booth and George S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *Journal of Computer and System Sciences (JCSS)*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

9 Mary Helen Briscoe. *A Researcher's Guide to Scientific and Medical Illustrations*. Springer Science & Business Media, 1990. doi:10.1007/978-1-4684-0355-8.

10 Thomas Depian, Martin Nöllenburg, Soeren Terziadis, and Markus Wallinger. Constrained Boundary Labeling, 2024. doi:10.48550/arXiv.2402.12245.

11 Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. Algorithms for Labeling Focus Regions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2583–2592, 2012. doi:10.1109/TVCG.2012.193.

12 Martin Fink and Subhash Suri. Boundary Labeling with Obstacles. In *Proc. 28th Canadian Conference on Computational Geometry (CCCG)*, pages 86–92. Simon Fraser University, 2016.

13 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

14 Stadtförsterei Fürth. Altersbestimmung und Baumanatomie, 2021. URL: https://www.stadtwald.fuerth.de/waldlehrpfad/baumanatomie-und-altersbestimmung. Accessed on 2024-04-16.

15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Sven Gedicke, Lukas Arzoumanidis, and Jan-Henrik Haunert. Automating the external placement of symbols for point features in situation maps for emergency response. *Cartography and Geographic Information Science (CaGIS)*, 50(4):385–402, 2023. doi:10.1080/15230406.2023.2213446.

17 Sven Gedicke, Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. Zoomless Maps: External Labeling Methods for the Interactive Exploration of Dense Point Sets at a Fixed Map Scale. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1247–1256, 2021. doi:10.1109/TVCG.2020.3030399.

18 Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. Multirow Boundary-Labeling Algorithms for Panorama Images. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 1(1):1:1–1:30, 2015. doi:10.1145/2794299.

19 Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Agent-Based Annotation of Interactive 3D Visualizations. In *Proc. 6th International Symposium on Smart Graphics (SG)*, volume 4073 of *Lecture Notes in Computer Science (LNCS)*, pages 24–35. Springer, 2006. doi:10.1007/11795018_3.

20 Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Contextual Grouping of Labels. In *Proc. 17th Simulation und Visualisierung (SimVis)*, pages 245–258. SCS Publishing House e.V., 2006. URL: http://www.simvis.org/Tagung2006/sv-proceedings.html.

21 Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. Multi-sided Boundary Labeling. *Algorithmica*, 76(1):225–258, 2016. doi:10.1007/s00453-015-0028-4.

22 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomás Vyskocil. Extending Partial Representations of Interval Graphs. *Algorithmica*, 78(3):945–967, 2017. doi:10.1007/s00453-016-0186-z.

23 Jonathan Klawitter, Felix Klesen, Joris Y. Scholl, Thomas C. van Dijk, and Alexander Zaft. Visualizing Geophylogenies - Internal and External Labeling with Phylogenetic Tree Constraints. In *Proc. 12th International Conference Geographic Information Science (GIScience)*, volume 277 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.GIScience.2023.5.

24 Konrad Mühler and Bernhard Preim. Automatic Textual Annotation for Surgical Planning. In *Proc. 14th International Symposium on Vision, Modeling, and Visualization (VMV)*, pages 277–284. DNB, 2009.

25 Benjamin Niedermann and Jan-Henrik Haunert. Focus+context map labeling with optimized clutter reduction. *International Journal of Cartography*, 5(2–3):158–177, 2019. doi:10.1080/23729333.2019.1613072.

26 Benjamin Niedermann, Martin Nöllenburg, and Ignaz Rutter. Radial Contour Labeling with Straight Leaders. In *Proc. 10th IEEE Pacific Visualization Symposium (PacificVis)*, PacificVis '17, pages 295–304. IEEE Computer Society, 2017. doi:10.1109/PACIFICVIS.2017.8031608.

27 Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, GIS '10, pages 310–319. Association for Computing Machinery (ACM), 2010. doi:10.1145/1869790.1869834.

28 Markus Tatzgern, Denis Kalkofen, and Dieter Schmalstieg. Dynamic Compact Visualizations for Augmented Reality. In *Proc. 20th IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 3–6. IEEE Computer Society, 2013. doi:10.1109/VR.2013.6549347.

29 Ian Vollick, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann. Specifying label layout style by example. In *Proc. 20th ACM Symposium on User Interface Software and Technology (UIST)*, UIST '07, pages 221–230. Association for Computing Machinery (ACM), 2007. doi:10.1145/1294211.1294252.

30 Colin Ware. *Chapter 6 – Static and Moving Patterns*, pages 179–237. Elsevier, 2013. doi:10.1016/b978-0-12-381464-7.00006-5.

Knapsack with Vertex Cover, Set Cover, and Hitting Set

Palash Dey

Indian Institute of Technology Kharagpur, India

Ashlesha Hota

Indian Institute of Technology Kharagpur, India

Sudeshna Kolay

Indian Institute of Technology Kharagpur, India

Sipra Singh

Indian Institute of Technology Kharagpur, India

Abstract

In the Vertex Cover Knapsack problem, we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with weights $(w(u))_{u\in\mathcal{V}}$ and values $(\alpha(u))_{u\in\mathcal{V}}$ of the vertices, the size s of the knapsack, a target value p, and the goal is to compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ with total weight at most s, and total value at least p. This problem simultaneously generalizes the classical vertex cover and knapsack problems. We show that this problem is strongly NP-complete. However, it admits a pseudo-polynomial time algorithm for trees. In fact, we show that there is an algorithm that runs in time $\mathcal{O}\left(2^{\text{tw}} \cdot n \cdot \min\{s^2, p^2\}\right)$ where tw is the treewidth of $\mathcal{G}$. Moreover, we can compute a $(1-\varepsilon)$-approximate solution for maximizing the value of the solution given the knapsack size as input in time $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ and a $(1+\varepsilon)$-approximate solution to minimize the size of the solution given a target value as input, in time $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ for every $\varepsilon > 0$. Restricting our attention to polynomial-time algorithms only, we then consider polynomial-time algorithms and present a 2 factor polynomial-time approximation algorithm for this problem for minimizing the total weight of the solution, which is optimal up to additive $o(1)$ assuming Unique Games Conjecture (UGC). On the other hand, we show that there is no ρ factor polynomial-time approximation algorithm for maximizing the total value of the solution given a knapsack size for any $\rho > 1$ unless P = NP.

Furthermore, we show similar results for the variants of the above problem when the solution $\mathcal{U}$ needs to be a minimal vertex cover, minimum vertex cover, and vertex cover of size at most k for some input integer k. Then, we consider set families (equivalently hypergraphs) and study the variants of the above problem when the solution needs to be a set cover and hitting set. We show that there are H_d and f factor polynomial-time approximation algorithms for Set Cover Knapsack where d is the maximum cardinality of any set and f is the maximum number of sets in the family where any element can belongs in the input for minimizing the weight of the knapsack given a target value, and a d factor polynomial-time approximation algorithm for d-Hitting Set Knapsack which are optimal up to additive $o(1)$ assuming UGC. On the other hand, we show that there is no ρ factor polynomial-time approximation algorithm for maximizing the total value of the solution given a knapsack size for any $\rho > 1$ unless P = NP for both Set Cover Knapsack and d-Hitting Set Knapsack.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Knapsack, vertex cover, minimal vertex cover, minimum vertex cover, hitting set, set cover, algorithm, approximation algorithm, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.27

Related Version *Full Version*: https://arxiv.org/abs/2406.01057 [8]

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 27; pp. 27:1–27:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *vertex cover* of an undirected graph is a set of vertices that contains at least one endpoint of every edge. For a real-world application of vertex cover, consider a city network $\mathcal{G}$ where the vertices are the major localities of the city, and we have an edge between two vertices if the distance between their corresponding locations is less than, say, five kilometers. A retail chain wants to open a few stores in the city in such a way that everyone (including the people living between any two major localities) in the city has a retail shop within five kilometers. The cost of opening a store depends on location. We can see that the company needs to compute a minimum weight vertex cover of the $\mathcal{G}$ to open stores with the minimum total cost, where the weight of a vertex is the cost of opening a store at that location. However, each store has the potential to generate non-core revenue, say from advertising. In such a scenario, the company may like to maximize the total non-core revenue without compromising its core business, which it will accomplish by opening stores at the vertices of a vertex cover. This is precisely what we call Vertex Cover Knapsack. In this problem, we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with weights $(w(u))_{u\in\mathcal{V}}$ and values $(\alpha(u))_{u\in\mathcal{V}}$ of the vertices, the size s of the knapsack, a target value p, and the goal is to compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ with $w(\mathcal{U}) = \sum_{u\in\mathcal{U}} w(u) \leqslant s$, and $\alpha(\mathcal{U}) = \sum_{u\in\mathcal{U}} \alpha(u) \geqslant p$.

We study several natural variations of this problem: (i) k-Vertex Cover Knapsack where the solution should be a vertex cover of size at most k for an integer input k, (ii) Minimal Vertex Cover Knapsack where the solution should be a minimal vertex cover, and (iii) Minimum Vertex Cover Knapsack where the solution should be a minimum vertex cover.

We then consider the hypergraphs or equivalently set families. There, we consider the knapsack generalization of the set cover and hitting set problems. In Set Cover Knapsack, we are given a collection $S_1, \ldots, S_m$ of subsets of a universe $[n]$, with weights $(w(j))_{j\in[m]}$ and values $(\alpha(j))_{j\in[m]}$ for the sets, the size s of the knapsack, a target value p, and the goal is to compute if there exists a set cover of total weight at most s and total value at least p. On the other hand, we have a collection $S_1, \ldots, S_m$ of d sized subsets of a universe $[n]$ with weights in d-Hitting Set Knapsack $(w(j))_{j\in[n]}$ and values $(\alpha(j))_{j\in[n]}$ for the elements, the size s of the knapsack, a target value p, and the goal is to compute if there exists a hitting set of total weight at most s and total value at least p.

1.1 Contributions

We study these problems under the lens of classical complexity theory, parameterized complexity, polynomial-time approximation, and FPT-approximation. We summarize our results in Table 1.

We now give a high-level overview of the techniques used in our results. For the f-approximation algorithm for Set Cover Knapsack, the dual LP of a configuration LP relaxation has two types of constraints: intuitively speaking, one set of constraints handles the knapsack part while the other set takes care of the set cover requirement. We first increase some dual variables iteratively so that some of the dual constraints corresponding to the knapsack part of the problem become tight. We pick the sets corresponding to these constraints. If this gives a valid set cover, then we are done. Otherwise, we increase some dual constraints iteratively corresponding to the set cover part of the problem until we satisfy the set cover requirements. The first part of our H_d-approximation algorithm is the same as the f-approximation algorithm. In the second part, we use the greedy algorithm for the set cover problem to pick more sets if the sets picked in the first part do not form a set cover.

Table 1 Summary of results. tw : treewidth of the graph, s : size of knapsack, p : target value of knapsack, ε : any real number greater than zero, n : number of vertices or size of the universe, f : the maximum number of sets where any element belongs, d : maximum size of any set, ρ : any poly-time computable function. $\star$: size of knapsack is input. † : bag size is input. ‡ : target value is input.

Knapsack variant	Results
Vertex Cover	• Strongly NP-complete (Observation 4) • NP-complete for star graphs (Observation 8) • Poly-time 2-approx. to minimize weight† (Corollary 17) • Poly-time ρ-inapprox. to maximize value* (Theorem 20) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$ (Theorem 21) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Vertex Cover of size $\leqslant k$	• Strongly NP-complete (Corollary 5) • NP-complete for star graphs (Observation 9) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$ (Theorem 23) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Minimum Vertex Cover	• NP hard (Observation 7) • NP-complete for trees (Observation 11) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$ (Theorem 22) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, , \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Minimal Vertex Cover	• Strongly NP-complete (Observation 6) • NP-complete for trees (Theorem 10) • No poly-time approx. algorithm neither to maximize value* nor to minimize weight† (Theorem 20) • $\mathcal{O}\left(16^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$ (Theorem 24) • $\mathcal{O}\left(16^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Theorem 25) • $\mathcal{O}\left(16^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Set Cover	• Strongly NP-complete (Observation 12) • Poly-time f-approx. to minimize weight† (Theorem 15) • Poly-time H_d-approx. to minimize weight† (Theorem 18) • Poly-time ρ-inapprox. to maximize value* (Theorem 20)
d-Hitting Set	• Strongly NP-complete (Observation 13) • Poly-time d-approx. to minimize weight† (Corollary 16) • Poly-time ρ-inapprox. to maximize value* (Theorem 20)

Our fixed-parameter pseudo-polynomial time algorithms with respect to treewidth for the variants of vertex cover knapsack combine the idea of pseudo-polynomial time algorithm and the dynamic programming algorithm for vertex cover with respect to treewidth. Then, we use these algorithms in a black-box fashion to obtain FPT-approximation algorithms.

1.2 Related Work

The classical knapsack problem admits a fully polynomial time approximation scheme (FPTAS) [25, 27]. Since our paper focuses on generalizations of knapsack to some graph theoretic problems and their extension to hypergraphs, we discuss only those knapsack variants directly related to ours.

Yamada et al. [28] proposed heuristics for knapsack when there is a graph on the items, and the solutions need to be an independent set. Many intractability results in special graph classes and heuristic algorithms based on pruning, dynamic programming, etc. have been studied for this independent set knapsack problem [18, 19, 23, 1, 17, 5, 21, 24, 14, 13, 2, 22]. Dey et al. [9] studied the knapsack problem with graph-theoretic constraints like - connectedness, paths, and shortest path.

Note that our Vertex Cover Knapsack also generalizes the classical weighted vertex cover problem, for which we know a polynomial-time 2-approximation algorithm which is the best possible approximation factor up to additive $\varepsilon > 0$ that one can achieve in polynomial time assuming Unique Games Conjecture [25, 27]. On the parameterized side, there is a long line of work on designing a fast FPT algorithm for vertex cover parameterized by the size k of a minimum vertex cover, with the current best being $\mathcal{O}\left(1.25284^k \cdot n^{\mathcal{O}(1)}\right)$ [16]. Later, Peter Damaschke [7] proved that it is solvable in time $\mathcal{O}\left(1.62^k \cdot n^{\mathcal{O}(1)}\right)$. Boria et al. [4] showed that there is a polynomial time $n^{-1/2}$ approximation algorithm and inapproximable within the ratio $n^{\varepsilon-1/2}$ in polynomial time unless $\mathsf{P} = \mathsf{NP}$, where $\varepsilon > 0$.

Various approximation algorithms have been studied for the Set Cover problem with approximation ratios f where f is the maximum number of sets that any element can belong and H_d where d is the maximum cardinality of any set, and H_d is the d-th harmonic number. These approximation factors are tight up to additive $\varepsilon > 0$ under standard complexity-theoretic assumptions [10, 20, 27, 25].

2 Preliminaries

We denote the set $\{1, 2, \ldots\}$ of natural numbers with $\mathbb{N}$. For any integer ℓ, we denote the sets $\{1, \ldots, \ell\}$ and $\{0, 1, \ldots, \ell\}$ by $[\ell]$ and $[\ell]_0$ respectively. We now define our problems formally. Our first problem is Vertex Cover Knapsack, where we need to find a vertex cover that fits the knapsack and meets a target value. A *vertex cover* of a graph is a subset of vertices that includes at least one end-point of every edge. Formally, it is defined as follows.

▶ **Definition 1** (Vertex Cover Knapsack). *Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, weight of vertices $(w(u))_{u\in\mathcal{V}}$, value of vertices $(\alpha(u))_{u\in\mathcal{V}}$, size s of knapsack, and target value p, compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ of $\mathcal{G}$ with weight $w(\mathcal{U}) = \sum_{u\in\mathcal{U}} w(u) \leqslant s$ and value $\alpha(\mathcal{U}) = \sum_{u\in\mathcal{U}} \alpha(u) \geqslant p$. We denote an any instance of it by $(\mathcal{G}, (w(u))_{u\in\mathcal{V}}, (\alpha(u))_{u\in\mathcal{V}}, s, p)$.*

The k-Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, Minimal Vertex Cover Knapsack problems are the same as Definition 1 except we require the solution $\mathcal{U}$ to be respectively a vertex cover of size at most k for an input integer k, a minimum cardinality vertex cover, a minimal vertex cover.

The treewidth of a graph quantifies the tree likeness of a graph [6]. Informally speaking, a tree decomposition of a graph is a tree where every node of the tree corresponds to some subsets of vertices, called bags, and the tree should satisfy three properties: (i) every vertex of the graph should belong to some bag, (ii) both the endpoints of every edge should belong to some bag, and (iii) the set of nodes of the tree containing any vertex should be connected. We refer to [8] for the formal definition of a tree decomposition, a nice tree decomposition, and the treewidth of a graph.

Extending the notion of vertex cover to hypergraphs, we define the Set Cover Knapsack problem where we need to compute a set cover that fits the knapsack and achieves a maximum value. Formally, we define it as follows.

▶ **Definition 2** (Set Cover Knapsack). *Given a collection $\mathcal{F} = \{S_1, \dots, S_m\}$ of subsets of a universe $[n]$ with weights $(w_j)_{j\in[m]}$ and values $(\alpha_j)_{j\in[m]}$ of the sets, size s of knapsack, and target value p, compute if there exists a set cover $\mathcal{J} \subseteq [m]$ of weight $w(\mathcal{J}) = \sum_{j\in\mathcal{J}} w_j \leqslant s$ and value $\alpha(\mathcal{J}) = \sum_{j\in\mathcal{J}} \alpha_j \geqslant p$. We denote any instance of it by $([n], \mathcal{F}, (w_j)_{j\in[m]}, (\alpha_j)_{j\in[m]}, s, p)$.*

We also define d-Hitting Set Knapsack, where we need to compute a hitting set that fits the knapsack and achieves at least the target value; here, items have weights and values.

▶ **Definition 3** (d-Hitting Set Knapsack). *Given a collection $\mathcal{F} = \{S_1, \dots, S_m\}$ of subsets of a universe $[n]$ of size at most d with weights $(w_i)_{i\in[n]}$ and values $(\alpha_i)_{i\in[n]}$ of the items, size s of knapsack, and target value p, compute if there exists a hitting set $\mathcal{I} \subseteq [n]$ of weight $w(\mathcal{I}) = \sum_{i\in\mathcal{I}} w_i \leqslant s$ and value $\alpha(\mathcal{I}) = \sum_{i\in\mathcal{I}} \alpha_i \geqslant p$. We denote any instance of it by $([n], \mathcal{S}, (w_i)_{i\in[n]}, (\alpha_i)_{i\in[n]}, s, p)$.*

If not mentioned otherwise, we use n to denote the number of vertices for problems involving graphs and the size of the universe for problems involving a set family; m to indicate the number of edges for problems involving graphs and the number of sets in the family of sets for problems involving a set family; tw to denote the treewidth of the graph; s to represent the size of knapsack, and p to denote the target value of solution.

3 Results: Classical NP Completeness

In this section, we present our NP-completeness results. Our first results show that Vertex Cover Knapsack is strongly NP-complete, that is, it is NP-complete even if the weight and value of every vertex are encoded in unary. We reduce from the classical Vertex Cover problem, where the goal is to find a vertex cover of cardinality at most some input integer k. Vertex Cover is known to be NP-complete even for 3 regular graphs [11, folklore]. To reduce a Vertex Cover instance to a Vertex Cover Knapsack instance, we define the weight and value of every vertex to be 1, and the size and target value to be k. In the interest of space, we omit the proofs of a few of our results. They are marked ($\star$). We refer to [8] for the detailed algorithm with proof of correctness and the analysis of its running time.

▶ **Observation 4.** Vertex Cover Knapsack *is strongly* NP*-complete.*

The same reduction in Observation 4 also shows that k-Vertex Cover Knapsack is strongly NP-complete.

▶ **Corollary 5.** k-Vertex Cover Knapsack *is strongly* NP*-complete.*

In the Maximum Minimal Vertex Cover problem, the goal is to compute if there exists a minimal vertex cover of cardinality at least some input integer. A vertex cover of a graph is called minimal if no proper subset of it is a vertex cover. Maximum Minimal Vertex Cover is known to be NP-complete [15, 3]. We show that the same reduction as in the proof of Observation 4 except starting from an instance of Maximum Minimal Vertex Cover instead of Vertex Cover, proves that Minimal Vertex Cover Knapsack is strongly NP-complete.

▶ **Observation 6** ($\star$). Minimal Vertex Cover Knapsack *is strongly* NP*-complete.*

We show similar results for Minimum Vertex Cover Knapsack except that it does not belong to NP unless the polynomial hierarchy collapses.

▶ **Observation 7** (⋆). Minimum Vertex Cover Knapsack *is strongly* NP*-hard.*

We show next that Vertex Cover Knapsack is NP-complete even if the underlying graph is a tree by reducing it from the classical Knapsack– simply add the knapsack items as leaves of a star graph. However, it turns out that they are not strongly NP-complete for trees. We will see in Section 5 that they admit pseudo-polynomial time algorithms for trees.

▶ **Observation 8** (⋆). Vertex Cover Knapsack *is* NP*-complete for star graphs.*

By setting k to be the number of leaves, the reduction in the proof of Observation 8 also shows NP-completeness for k-Vertex Cover Knapsack.

▶ **Observation 9** (⋆). k-Vertex Cover Knapsack *is* NP*-complete for star graphs.*

Note that the reduction from Knapsack to Vertex Cover Knapsack for star graphs does not work for Minimal Vertex Cover Knapsack and Minimum Vertex Cover Knapsack. Indeed, for star graphs, both the problems admit polynomial-time algorithms. Nevertheless, we are able to show that both the problems are (not strongly) NP-complete for trees.

▶ **Theorem 10** (⋆). Minimal Vertex Cover Knapsack *is* NP*-complete for trees.*

▶ **Observation 11** (⋆). Minimum Vertex Cover Knapsack *is* NP*-complete for trees.*

Note that, since the size of a minimum vertex cover in a tree can be computed in polynomial time thanks to König's Theorem [26], Minimum Vertex Cover Knapsack belongs to NP.

We show similar results for Set Cover Knapsack and d-Hitting Set Knapsack also by reducing from respectively unweighted set cover and unweighted d-hitting set, both of which are known to be NP-complete [12].

▶ **Observation 12** (⋆). Set Cover Knapsack *is strongly* NP*-complete.*

▶ **Observation 13** (⋆). d-Hitting Set Knapsack *is strongly* NP*-complete.*

4 Results: Polynomial Time Approximation Algorithms

In this section, we focus on the polynomial-time approximability of our problems. For all the problems in this paper, we study two natural optimization versions: (i) minimizing the weight of the solution given a target value as input and (ii) maximizing the value of the solution given knapsack size as input. We first consider minimizing the weight of the solution.

A natural integer linear programming formulation of Vertex Cover Knapsack is the following.

minimize $\sum_{u\in\mathcal{V}} w(u)x_u$

Subject to:

$$\begin{aligned} & x_u + x_v \geqslant 1, \forall (u,v) \in \mathcal{E} \\ & \sum_{u\in\mathcal{V}} \alpha(u)x_u \geqslant p \\ & x_u \in \{0,1\}, \forall u \in \mathcal{V} \end{aligned} \tag{1}$$

We replace the constraints $x_u \in \{0,1\}$, with $x_u \geqslant 0$, $\forall u \in \mathcal{V}$ to obtain linear programming (abbreviated as LP) relaxation of the integer linear program (abbreviated as ILP).

▶ **Observation 14.** *The relaxed LP of the ILP 1 has an unbounded integrality gap. To see this, consider an edgeless graph on two vertices v_1 and v_2. Let $w(v_1) = 0$, $w(v_2) = 1$, $\alpha(v_1) = p - 1$ and $\alpha(v_2) = p$. The optimal solution to ILP sets $x_{v_1} = 0, x_{v_2} = 1$, for a total weight of 1. However, the optimal solution to the relaxed LP sets $x_{v_1} = 1, x_{v_2} = 1/p$ and has a total weight of $1/p$. Thus, in this case, the integrality gap is at least $\frac{1}{1/p} = p$.*

To tackle Observation 14, we strengthen the inequality $\sum_{u \in \mathcal{V}} \alpha(u)x_u \geqslant p$. This allows us to obtain an f approximation algorithm even for the more general Set Cover Knapsack problem that we present now. In particular, in addition to having a constraint for every element of the universe, we have a constraint for every $\mathcal{A} \subseteq \mathcal{F}$ of sets such that $\alpha(A) = \sum_{i \in \mathcal{A}} \alpha(i) < p$ where p is the target value given as input. We define the residual value $p_{\mathcal{A}} = p - \alpha(\mathcal{A})$. Given the set $\mathcal{A}$, we simplify the problem on the sets $\mathcal{F} - \mathcal{A}$, where the target value is now $p_{\mathcal{A}}$. We also reduce the value of each set $S_i \in \mathcal{F} - \mathcal{A}$ to be the minimum of its own value and $p_{\mathcal{A}}$, i.e., let $\alpha^{\mathcal{A}}(i) = \min(\alpha(i), p_{\mathcal{A}})$. We can now give the following Integer linear programming formulation of the problem:

minimize $\sum_{i \in [m]} w(i)x_i$

Subject to:

$$\sum_{i : e_j \in \mathcal{S}_i} x_i \geqslant 1, \forall e_j \in \mathcal{U}$$
$$\sum_{i \in \mathcal{F} - \mathcal{A}} \alpha^{\mathcal{A}}(i)x_i \geqslant p_{\mathcal{A}}, \forall \mathcal{A} \subseteq \mathcal{F}$$
$$x_i \in \{0, 1\}, \forall i \in [m]$$

We replace the constraints $x_i \in \{0, 1\}$, with $x_i \geqslant 0$ to obtain the LP relaxation of the ILP. The dual of the LP relaxation is :

maximize $\sum_{\mathcal{A} : \mathcal{A} \subseteq \mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + \sum_{j \in [n]} y_j$

Subject to:

$$\sum_{j : e_j \in \mathcal{S}_i} y_j \leqslant w(i), \forall \mathcal{S}_i \in \mathcal{F}$$
$$\sum_{\mathcal{A} \subseteq \mathcal{F} : i \notin \mathcal{A}} \alpha^{\mathcal{A}}(i) y_{\mathcal{A}} \leqslant w(i), \forall i \in \mathcal{F}$$
$$y_{\mathcal{A}} \geqslant 0, \forall \mathcal{A} \subset \mathcal{F}$$

In our primal-dual algorithm, we begin with dual feasible solution $y = 0$ and partial solution $\mathcal{A} = \emptyset$. We pick one set in every iteration until the value of the set $\mathcal{A}$ of sets becomes at least the target value p. We increase the dual variable $y_{\mathcal{A}}$ in every iteration until the dual constraint for some set $i \in \mathcal{F} - \mathcal{A}$ becomes tight. We then pick that set in our solution and continue. After this loop terminates, the value of the set $\mathcal{A}$ of sets is at least the target value p. At that point, if $\mathcal{A}$ is a set cover, then we output $\mathcal{A}$. Otherwise, till there exists an element e_j of the universe that is not covered by $\mathcal{A}$, we increase the dual variable y_j until the dual constraint for some set ℓ with $e_j \in S_l$ becomes tight. We then include S_ℓ in $\mathcal{A}$ and continue. We present the pseudocode of our algorithm in Algorithm 1.

▶ **Theorem 15.** *Algorithm 1 is an f-approximation algorithm for the* Set Cover Knapsack *problem for minimizing the weight of the solution, where f is the maximum number of sets in the family where any element belongs.*

Algorithm 1 Primal-dual f-approximation algorithm for Set Cover Knapsack.

```
$y \leftarrow 0, \mathcal{A} \leftarrow \emptyset$
while $\alpha(\mathcal{A}) < p$ do
    Increase $y_{\mathcal{A}}$ until for some $i \in \mathcal{F} - \mathcal{A}$, $\sum_{\mathcal{B} \subseteq \mathcal{F} : i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = w(i)$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
end while
$\mathcal{X} \leftarrow \mathcal{A}, \mathcal{A}' \leftarrow \mathcal{A}$
while $\exists e_j \notin \bigcup_{i \in \mathcal{A}'} S_i$ do
    Increase $y_j$ until there is some $t$ with $e_j \in S_t$ such that $\sum_{j : e_j \in S_t} y_j = w(t)$
    $\mathcal{A}' \leftarrow \mathcal{A} \cup \{t\}$
end while
return $\mathcal{A}'$
```

Proof. Let ALG be the weight of the set cover $\mathcal{A}'$ output by Algorithm 1. Then

$$\begin{aligned}\mathsf{ALG} &= \sum_{i \in \mathcal{A}'} w(i) x_i \\ &= \sum_{i \in \mathcal{X}} w(i) x_i + \sum_{i \in \mathcal{A}' - \mathcal{X}} w(i) x_i\end{aligned}$$

Let OPT be the optimal weight of the Set Cover Knapsack instance, set i picked in the i-th iteration of the first while loop (which we can assume without loss of generality by renaming the sets), and l the set selected by the algorithm at the last iteration of the first while loop. Since the first while loop terminates when $\alpha(\mathcal{A}) \geqslant p$, we know that $\alpha(\mathcal{X}) \geqslant p$; since set l was added to $\mathcal{X}$, it must be the case that before l was added, the total value of $\mathcal{A}$ was less than p, so that $\alpha(\mathcal{X} - \{l\}) < p$. For $i \in [l]$, we define $\mathcal{A}_i = [i]$ as the set of sets picked in the first i iterations of the first while loop and $\mathcal{C} = \{\mathcal{A}_i : i \in [l]\}$. We observe that a dual variable $y_{\mathcal{B}}$ is non-zero only if $\mathcal{B} \in \mathcal{C}$. Since we pick only tight sets, we have

$$\sum_{i \in \mathcal{X}} w(i) = \sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \subseteq \mathcal{F} : i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = \sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \in \mathcal{C} : i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}}.$$

Reversing the double sum, we have

$$\sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \in \mathcal{C} : i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = \sum_{\mathcal{B} \in \mathcal{C}} y_{\mathcal{B}} \sum_{i \in \mathcal{X} - \mathcal{B}} \alpha^{\mathcal{B}}(i).$$

Note that in any iteration of the algorithm except the last one, adding the next set i to the current sets in $\mathcal{A}$ did not cause the value of the knapsack to become at least p; that is, $\alpha(i) < p - \alpha(\mathcal{A}) = p_{\mathcal{A}}$ at that point in the algorithm. Thus, for all sets $i \in \mathcal{A}$ except l, $\alpha^{\mathcal{A}}(i) = \min(\alpha(i), p_{\mathcal{A}}) = \alpha(i)$, for the point in the algorithm at which $\mathcal{A}$ was the current set of sets. Thus, we can rewrite

$$\sum_{i \in \mathcal{X} - \mathcal{A}} \alpha^{\mathcal{A}}(i) = \alpha^{\mathcal{A}}(l) + \sum_{i \in \mathcal{X} - \mathcal{A} : i \neq l} \alpha^{\mathcal{A}}(i) = \alpha^{\mathcal{A}}(l) + \alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}).$$

Note that $\alpha^{\mathcal{A}}(l) \leqslant p_{\mathcal{A}}$ by definition, and as argued at the beginning of the proof $\alpha(\mathcal{X} - \{l\}) < p$ so that $\alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}) < p - \alpha(\mathcal{A})) = p_{\mathcal{A}}$; thus, we have that

$$\alpha^{\mathcal{A}}(l) + \alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}) < 2p_{\mathcal{A}}$$

which is the same as saying

$$\sum_{i\in\mathcal{X}-\mathcal{B}} \alpha^{\mathcal{B}}(i) < 2p_{\mathcal{B}} \text{ for every } \mathcal{B}\in\mathcal{C}.$$

Therefore,

$$\sum_{i\in\mathcal{X}} w(i) = \sum_{\mathcal{B}\in\mathcal{C}} y_{\mathcal{B}} \sum_{i\in\mathcal{X}-\mathcal{B}} \alpha^{\mathcal{B}}(i) < 2\sum_{\mathcal{B}:\mathcal{B}\in\mathcal{C}} p_{\mathcal{B}}y_{\mathcal{B}} = 2\sum_{\mathcal{B}\subseteq\mathcal{F}:i\notin\mathcal{B}} p_{\mathcal{B}}y_{\mathcal{B}}$$

where the last equality follows from the fact that $y_{\mathcal{B}}=0$ if $\mathcal{B}\notin\mathcal{C}$.

Our algorithm picks sets in $\mathcal{A}\prime\setminus\mathcal{X}$ in the second while loop if the set of sets picked in the first while loop does not form a set cover. We now upper bound $\sum_{i\in\mathcal{A}'\setminus\mathcal{X}} w(i)$ as follows.

$$\sum_{i\in\mathcal{A}'\setminus\mathcal{X}} w(i) = \sum_{i\in\mathcal{A}'\setminus\mathcal{X}}\ \sum_{j\in[n]:e_j\in S_i} y_j = \sum_{j\in[n]} |\{i\in\mathcal{A}'\setminus\mathcal{X} : e_j\in S_i\}|y_j \leqslant f\sum_{j\in[n]} y_j$$

The first equality follows from the fact that only tight sets are picked. We now bound ALG.

$$\begin{aligned}
\mathsf{ALG} &= \sum_{i\in\mathcal{A}'} w(i)x_i\\
&= \sum_{i\in\mathcal{X}} w(i)x_i + \sum_{i\in\mathcal{A}'-\mathcal{X}} w(i)x_i\\
&\leqslant 2\sum_{A:A\subseteq I} p_{\mathcal{A}}y_{\mathcal{A}} + f\sum_{j\in[n]} y_j\\
&\leqslant f\left(\sum_{A:A\subseteq I} p_{\mathcal{A}}y_{\mathcal{A}} + \sum_{j\in[n]} y_j\right)\\
&= f\mathsf{OPT}
\end{aligned}$$ ◀

We note that our algorithm is a combinatorial algorithm based on the primal-dual framework – in particular, we use LPs only to design and analyze our algorithm. We do not need to solve any LP. We obtain approximation algorithms for the Vertex Cover Knapsack and d-Hitting Set Knapsack problems as corollaries of Theorem 15 by reducing these problems to Set Cover Knapsack.

▶ **Corollary 16** (⋆). *There exists a* $\boldsymbol{d}$*-approximation algorithm for* d*-*Hitting Set Knapsack *for minimizing the weight of the solution. The algorithm is combinatorial in nature and based on the primal-dual method.*

▶ **Corollary 17** (⋆). *There exists a* 2*-approximation algorithm for* Vertex Cover Knapsack *for minimizing the weight of the solution. The algorithm is combinatorial in nature and based on the primal-dual method.*

We next present a H_d-approximation algorithm for Set Cover Knapsack where d is the maximum cardinality of any set in the input instance and $H_d=\sum_{i=1}^{d}\frac{1}{i}$ is the d-th harmonic number. The idea is to run the first while loop of Algorithm 1, and then, if the selected sets do not cover the universe, then, instead of the second while loop of Algorithm 1, we pick sets following the standard greedy algorithm for minimum weight set cover. We show that the algorithm achieves an approximation factor of $\max(2,H_d)$ by analyzing it using the *dual fitting technique.*

Algorithm 2 Max(2, H_d)-approximation algorithm for Set Cover Knapsack.

```
$y \leftarrow 0, \mathcal{A} \leftarrow \emptyset$
while $\alpha(\mathcal{A}) < p$ do
    Increase $y_{\mathcal{A}}$ until for some $i \in \mathcal{F} - \mathcal{A}, \sum_{\mathcal{B} \subseteq \mathcal{F} : i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = w(i)$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
end while
$\mathcal{X} \leftarrow \mathcal{A}, \mathcal{U}' \leftarrow \mathcal{U} - \bigcup_{i \in \mathcal{X}} \mathcal{S}_i, \mathcal{F}' \leftarrow \mathcal{F} - \mathcal{X}, I \leftarrow \emptyset, \hat{S}_i \leftarrow S_i$ for all $i \in \mathcal{F}'$
while $I$ is not a set cover for $\mathcal{U}'$ do
    $l \leftarrow \arg\min_{i : \hat{S}_i \neq \emptyset} \frac{w(i)}{|\hat{S}_i|}$
    $I \leftarrow I \cup \{l\}$
    $\hat{S}_i \leftarrow \hat{S}_i - S_l$ for all $i \in \mathcal{F}'$
end while
return $\mathcal{X} \cup \mathcal{I}$
```

▶ **Theorem 18.** *Algorithm 2 is a max(2, H_d)-approximation algorithm for the* Set Cover Knapsack *problem for minimizing the weight of the solution, where d is the maximum cardinality of any set in the input.*

Proof. We follow the same notation defined in Algorithm 2 in this proof. Since the first part of Algorithm 2 is the same as the first part of Algorithm 1, from the proof of Theorem 15, we have

$$\sum_{i \in \mathcal{X}} w(i) < 2 \sum_{\mathcal{B} \subseteq \mathcal{F} : i \notin \mathcal{B}} p_{\mathcal{B}} y_{\mathcal{B}}.$$

To bound the sum of weights of the sets in $\mathcal{I}$, we use the dual fitting technique. In particular, we will first construct an assignment of dual variables $(y_j)_{j \in [n]}$ with $\sum_{i \in \mathcal{I}} w_i = \sum_{j=1}^{n} y_j$. However, $(y_j)_{j \in [n]}$ may not satisfy the dual constraints involving those variables. However, and then show that $y'_j = \frac{1}{H_d} y_j, j \in [n]$ satisfies all dual constraints involving those variables. We concretize this idea below.

Whenever Algorithm 2 includes a set $\hat{\mathcal{S}}_i$ in $\mathcal{I}$, we define $y_j = \frac{w(i)}{|\hat{\mathcal{S}}_i|}$ for each $j \in \hat{\mathcal{S}}_i$. Since each $j \in \hat{\mathcal{S}}_i$ is uncovered in iteration when Algorithm 2 picks the set $\hat{\mathcal{S}}_i$, and is then covered for the remaining iterations of the algorithm (because we added subset $\mathcal{S}_i$ to $\mathcal{I}$), the dual variable y_j is set to a value exactly once. Furthermore, we see that

$$w(i) = \sum_{i : j \in \hat{\mathcal{S}}_i} y_j, \forall i \in \mathcal{I}$$

since the weight $w(i)$ of the set i is distributed among $y_j, j \in \hat{\mathcal{S}}_i$. Hence, we have,

$$\sum_{j \in \mathcal{I}} w(i) = \sum_{i=1}^{n} y_j.$$

We claim that $y'_j = \frac{1}{H_d} y_j$ for all $j \in [n]$ satisfies the dual constraints involving these variables. For that, we need to show that for each subset $\mathcal{S}_i, i \in [m]$,

$$\sum_{i : j \in \mathcal{S}_i} y'_j \leqslant w(i).$$

Pick an arbitrary subset $\mathcal{S}_i$ and an arbitrary iteration k of the second while loop of Algorithm 2. Let ℓ be the number of iterations that the second while loop of Algorithm 2 makes and a_k the number of elements in this subset that is still uncovered at the beginning of the k-th iteration, so that $a_1 = |\mathcal{S}_i|$, and $a_{\ell+1} = 0$. Let A_k be the set of uncovered elements of $\mathcal{S}_i$ covered in the k-th iteration, so that $|A_k| = a_k - a_{k+1}$. If subset $\mathcal{S}_q$ is chosen in the k-th iteration, then for each element $j \in A_k$ covered in the k-th iteration, we have

$$y'_j = \frac{w_q}{H_d|\hat{\mathcal{S}}_q|} \leqslant \frac{w(i)}{H_d a_k},$$

where $\hat{\mathcal{S}}_q$ is the set of uncovered elements of $\mathcal{S}_q$ at the beginning of the k-th iteration. The inequality follows because if $\mathcal{S}_q$ is chosen in the k-th iteration, it must minimize the ratio of its weight to the number of uncovered elements it contains. Thus,

$$\begin{aligned}
\sum_{i:e_j\in\mathcal{S}_i} y'_j &= \sum_{k=1}^{l} \sum_{j\in[n]:j\in A_k} y'_j \\
&\leqslant \sum_{k=1}^{l} (a_k - a_{k+1}) \frac{w(i)}{H_d a_k} \\
&\leqslant \frac{w(i)}{H_d} \sum_{k=1}^{l} \frac{a_k - a_{k+1}}{a_k} \\
&\leqslant \frac{w(i)}{H_d} \sum_{k=1}^{l} \left(\frac{1}{a_k} + \frac{1}{a_k - 1} + \cdots + \frac{1}{a_{k+1}+1} \right) \\
&\leqslant \frac{w(i)}{H_d} \sum_{i=1}^{|\mathcal{S}_i|} \frac{1}{i} \\
&= \frac{w(i)}{H_d} H_{|\mathcal{S}_i|} \\
&\leqslant w(i),
\end{aligned}$$

where the final inequality follows because $|\mathcal{S}_i| \leqslant d$. Hence, $((y_{\mathcal{B}})_{\mathcal{B}\in\mathcal{F}}, (y'_j)_{j\in[n]})$ is a dual feasible solution. We now bound ALG as follows.

$$\begin{aligned}
\mathsf{ALG} &= \sum_{i\in\mathcal{X}} w(i)x_i + \sum_{i\in\mathcal{I}} w(i)x_i \\
&\leqslant 2 \sum_{A:A\subseteq\mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + H_d \sum_{j\in[n]} y_j \\
&= \max(2, H_d) \left(\sum_{A:A\subseteq\mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + \sum_{j\in[n]} y_j \right) \\
&= \max(2, H_d) \cdot \mathsf{OPT}
\end{aligned}$$
◀

The approximation guarantees of Theorems 15 and 18 are the best possible approximation guarantees, up to any additive constant $\varepsilon > 0$, that any polynomial time algorithm hopes to achieve, assuming standard complexity-theoretic assumptions.

▶ **Theorem 19** (⋆). *Let $\varepsilon > 0$ be any constant. Then we have the following:*

1. *There is no polynomial-time $(1 - \varepsilon)\ln n$ factor approximation algorithm for* Set Cover Knapsack *unless every problem in* NP *admits a quasi-polynomial time algorithm.*
2. *Assuming Unique Games Conjecture (UGC), there is no polynomial-time $(1 - f)\ln n$ factor approximation algorithm for* Set Cover Knapsack.

3. *Assuming Unique Games Conjecture (UGC), there is no polynomial-time* $(1-d)\ln n$ *factor approximation algorithm for d*-Hitting Set Knapsack.

We next focus on maximizing the value of the solution given a knapsack size as input. Surprisingly, for all the problems studied in this paper, we show that there is no ρ-approximation algorithm for any of our problems for any $\rho > 1$.

▶ **Theorem 20** ($\star$). *For any $\rho > 1$, there does not exist a ρ-approximation algorithm for maximizing the value of the solution given the size of the knapsack for* Set Cover Knapsack, *d*-Hitting Set Knapsack, Vertex Cover Knapsack, Minimal Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, *and k*-Vertex Cover Knapsack *unless* P = NP.

The inapproximability barriers of Theorems 19 and 20 can be overcome using the framework of FPT-approximation. In particular, we will show FPT $(1-\varepsilon)$-approximation algorithms, parameterized by the treewidth of the input graph, for all four variants of vertex cover knapsack for maximizing the value of the solution.

5 Results: Parameterized Complexity

We study the four variants of Vertex Cover Knapsack using the framework of parameterized complexity. For that, we consider the treewidth of the input graph as a parameter. With respect to treewidth, we design algorithms that run in time single exponential in the treewidth times polynomial in n (number of vertices), size s, and target value p of the knapsack. We then use these algorithms to develop a $(1-\varepsilon)$-approximation algorithm for maximizing the value of the solution that runs in time single exponential in the treewidth times polynomial in the number n of vertices, $1/\varepsilon$ and $\sum_{v\in\mathcal{V}} \alpha(v)$.

We know that there exists a $\mathcal{O}\left(2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot n\right)$ time algorithm for the Vertex Cover problem [6]. It turns out that it is relatively easy to modify that algorithm to design a similar algorithm Vertex Cover Knapsack, k-Vertex Cover Knapsack, and Minimum Vertex Cover Knapsack.

▶ **Theorem 21** ($\star$). *There is an algorithm for* Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$.

It turns out that the main idea of the algorithm of Theorem 21 can be modified to obtain algorithms for Minimum Vertex Cover Knapsack and k-Vertex Cover Knapsack with similar running times.

▶ **Theorem 22** ($\star$). *There is an algorithm for* Minimum Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$.

▶ **Theorem 23** ($\star$). *There is an algorithm for k*-Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$.

However, the approach of Theorem 21 breaks down for Minimal Vertex Cover Knapsack. This is so because a minimal vertex cover (unlike a vertex cover, a vertex cover of size at most k, and a minimum vertex cover) of a graph may not be a minimal vertex cover of some of its induced subgraphs. For this reason, it is not enough to keep track of all minimal vertex covers of the subgraphs rooted at some tree node intersecting the bag at certain subsets. Intuitively speaking, we tackle this problem by adding another subset of vertices in the "indices" of the DP table that will be part of some minimal vertex cover of some other induced subgraph.

▶ **Theorem 24.** *There is an algorithm for* Minimal Vertex Cover Knapsack *with running time* $\mathcal{O}\left(16^{tw} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$.

Proof. Let $(G = (V, E), (w(u))_{u \in V}, (\alpha(u))_{u \in V}, s, p)$ be an input instance of Minimal Vertex Cover Knapsack and $(\mathbb{T} = (V_{\mathbb{T}}, E_{\mathbb{T}}), \mathcal{X})$ a nice tree decomposition rotted at node r of treewidth $\text{tw}(G)$.

We define a function $\ell : V_{\mathbb{T}} \to \mathbb{N}$. For a vertex $t \in V_{\mathbb{T}}$, $\ell(t) = \mathsf{dist}_{\mathbb{T}}(\mathsf{t}, \mathsf{r})$, where r is the root. Note that this implies that $\ell(r) = 0$. Let us assume that the values that ℓ take over the nodes of $\mathbb{T}$ are between 0 and L. For a node $t \in V_{\mathbb{T}}$, we denote the set of vertices in the bags in the subtree rooted at t by V_t and $G_t = G[V_t]$. We now describe a dynamic programming algorithm over $(\mathbb{T}, \mathcal{X})$ for Minimal Vertex Cover Knapsack.

States. We maintain a DP table D where a state has the following components:

1. t represents a node in $V_{\mathbb{T}}$.
2. V_1, V_2 are subsets of the bag X_t, not necessarily disjoint.
3. V_1 represents the intersection of X_t with a minimal vertex cover of the subgraph $G_t[(V_t \setminus V_2) \cup V_1]$.

Interpretation of States. For each node $t \in \mathbb{T}, V_1, V_2 \subseteq X_t$ and "undominated" minimal vertex cover S of the induced graph $G_t[(V_t \setminus V_2) \cup V_1]$ such that $S \cap X_t = V_1$, we store $(w(S), \alpha(S))$ in the list $D[t, V_1, V_2]$. We say a minimal vertex cover $S_1 \subseteq (V_t \setminus V_2) \cup V_1$ dominates another minimal vertex cover $S_2 \subseteq (V_t \setminus V_2) \cup V_1$ if $w(S_1) \leqslant w(S_2)$ and $\alpha(S_1) \geqslant \alpha(S_2)$ with at least one inequality being strict. We say a minimal vertex cover of $G_t[(V_t \setminus V_2) \cup V_1]$ *undominated* if no other minimal vertex cover of $G_t[(V_t \setminus V_2) \cup V_1]$ dominates it.

For each state $D[t, V_1, V_2]$, we initialize $D[t, V_1, V_2]$ to the list $\{(0, 0)\}$.

Dynamic Programming on D. We first update the table for states with nodes $t \in V_{\mathbb{T}}$ such that $\ell(t) = L$. When all such states are updated, then we update states where the level of node t is $L - 1$, and so on, till we finally update states with r as the node – note that $\ell(r) = 0$. For a particular j, $0 \leqslant j < L$ and a state $[t, V_1, V_2]$ such that $\ell(t) = j$, we can assume that $D[t, V_1, V_2]$ have been evaluated for all t', such that $\ell(t') > j$ and all subsets V_1' and V_2' of $X_{t'}$. Now we consider several cases by which $D[t, V_1, V_2]$ is updated based on the nature of t in $\mathbb{T}$:

1. Suppose t is a leaf node with $X_t = \{v\}$. Then $D[t, v, \emptyset] = (w(v), \alpha(v))$, or $D[t, \emptyset, v] = (0, 0)$ and $D[t, \emptyset, \emptyset]$ stores the pair $(0, 0)$.
2. Suppose t is an introduce node. Then it has an only child t' where $X_{t'} \cup \{u\} = X_t$. Then for all $\mathbb{S} \subseteq X_t$: If $\mathbb{S}$ is not a vertex cover of $G[X_t]$, we set $D[t, V_1, V_2] = (\infty, \infty)$. Otherwise, we have three cases:
   a. Case 1: If $u \notin V_1 \cup V_2$, then we copy each pair (w, α) from $D[t', V_1, V_2]$
   b. Case 2: If $u \in V_1 \setminus V_2$, then
      i. we check if $N(u) \setminus V_1 \neq \emptyset$, then
         A. for each pair (w, α) in $D[t', V_1 \setminus \{u\}, V_2]$, if $w + w(u) \leqslant s$, then we put $(w + w(u), \alpha + \alpha(u))$ in $D[t', V_1 \setminus \{u\}, V_2]$ to $D[t, V_1, V_2]$.
         B. for each pair (w, α) in $D[t', V_1 \setminus \{u\}, V_2]$, if $w + w(u) > s$, we put (w, α) to $D[t, V_1, V_2]$.
      ii. Otherwise we store (∞, ∞).
   c. Case 3: If $u \in V_2$, then we set $D[t, V_1, V_2] = D[t', V_1, V_2 \setminus \{u\}]$.
3. Suppose t is a forget vertex node. Then it has an only child t' where $X_t \cup \{u\} = X_{t'}$. We copy all the pairs from $D[t', V_1 \cup \{u\}, V_2]$, $D[t', V_1, V_2 \cup \{u\}]$ and $D[t', V_1, V_2]$ to $D[t, V_1, V_2]$ and remove all dominated pairs.

4. Suppose t is a join node. Then it has two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let $(w(V_1 \cap V_2), \alpha(V_1 \cap V_2))$ be the total weight and value of the vertices in $V_1 \cap V_2$. Then for all $W_1, W_2 \subseteq V_1 \subseteq X_t$, consider a pair (w_1, α_1) in $D[t_1, W_1, W_2 \cup V_2]$ and a pair (w_2, α_2) in $D[t_2, W_2, W_1 \cup V_2]$. Suppose $w_1 + w_2 - w(V_1 \cap V_2) \leqslant s$. Then we add $(w_1 + w_2 - w(V_1 \cap V_2), \alpha_1 + \alpha_2 - \alpha(V_1 \cap V_2))$ to $D[t, V_1, V_2]$.

Finally, in the last step of updating $D[t, V_1, V_2]$, we go through the list saved in $D[t, V_1, V_2]$ and only keep undominated pairs.

Correctness of the algorithm. Recall that we are looking for a solution $\mathcal{U}$ that contains the fixed vertex v that belongs to all bags of the tree decomposition. In each state we maintain the invariant $V_1, V_2 \subseteq X_t$ such that $V_1 = X_t \cap$ minimal vertex cover knapsack of $G_t \setminus$ edges incident on $V_2 \setminus V_1$. First, we show that a pair (w, α) belonging to $D[t, V_1, V_2]$ for a node $t \in V_{\mathbb{T}}$ and a subset $\mathbb{S}$ of X_t corresponds to a minimal vertex cover knapsack H in G_t. Recall that $X_r = \{v\}$. Thus, this implies that a pair (w, α) belonging to $D[r, V_1 = \{v\}, V_2 = \emptyset)]$ corresponds to a minimal vertex cover knapsack of G. Moreover, the output is a pair that is feasible and with the highest value.

In order to show that a pair (w, α) belonging to $D[t, V_1, V_2]$ for a node $t \in V_{\mathbb{T}}$ and a subset V_1 of X_t corresponds to a minimal vertex cover knapsack of $G_t \setminus$ edges incident on $V_2 \setminus V_1$, we need to consider the cases of what t can be:

1. **Leaf node**: Recall that in our modified nice tree decomposition we have added a vertex v to all the bags. Suppose a leaf node t contains a single vertex v, $D[t, v, \emptyset] = (w(v), \alpha(v))$, $D[t, \emptyset, v] = (0, 0)$ and $D[L, \emptyset, \emptyset]$ stores the pair $(0, 0)$. This is true in particular when $j = L$, the base case. From now we can assume that for a node t with $\ell(t) = j < L$ and all subsets $V_1, V_2 \subseteq X_t$, $D[t', V_1'', V_2'']$ entries are correct and correspond to minimal vertex cover in $G_{t'} \setminus$ edges incident on $V_2'' \setminus V_1''$. when $\ell(t') > j$.
2. **Introduce node**: When t is an introduce node, there is a child t'. We are introducing a vertex u and the edges associated with it in G_t. Since $\ell(t') > \ell(t)$, by induction hypothesis all entries in $D[t', V_1'' = V_1 \setminus \{u\}, V_2'' = V_2 \setminus \{u\}]$, $D[t', V_1'' = V_1 \setminus \{u\}, V_2'' = V_2]$, and $D[t', V_1'' = V_1, V_2'' = V_2 \setminus \{u\}]$, $\forall\ V_1'', V_2'' \subseteq X_{t'}$ are already computed and feasible. We update pairs in $D[t, V_1, V_2]$ from $D[t', V_1 \setminus \{u\}, V_2]$ or $D[t', V_1, V_2 \setminus \{u\}]$ or $D[t', V_1 \setminus \{u\}, V_2 \setminus \{u\}]$ such that either u is considered as part of a minimal solution in $G_t \setminus$ edges incident on $V_2 \setminus V_1$ or not.
3. **Forget node**: When t is a forget node, there is a child t'. We are forgetting a vertex u and the edges associated with it in G_t. Since $\ell(t') > \ell(t)$, by induction hypothesis all entries in $D[t', V_1'' = V_1 \cup \{u\}, V_2'' = V_2]$, $D[t', V_1'' = V_1, V_2'' = V_2 \cup \{u\}]$, and $D[t', V_1'' = V_1, V_2'' = V_2]$, $\forall\ V_1'', V_2'' \subseteq X_{t'}$ are already computed and feasible. We copy each undominated (w, α) pair stored in $D[t', V_1 \cup \{u\}, V_2]$, $D[t', V_1, V_2 \cup \{u\}]$ and $D[t', V_1, V_2]$ to $D[t, V_1, V_2]$.
4. **Join node**: When t is a join node, there are two children t_1 and t_2 of t, such that $X_t = X_{t_1} = X_{t_2}$. For all subsets $V_1 \subseteq X_t$ we partition V_1 into two subsets W_1 and W_2 (not necessarily disjoint) such that W_1 is the intersection of X_{t_1} with minimal solution in the graph $G_{t_1} \setminus$ edges incident on $(W_2 \cup V_2) \setminus W_1$. Similarly, W_2 is the intersection of X_{t_2} with minimal solution in the graph $G_{t_2} \setminus$ edges incident on $(W_1 \cup V_2) \setminus W_2$. By the induction hypothesis, the computed entries in $D[t_1, W_1, W_2 \cup V_2]$ and $D[t_2, W_2, W_1 \cup V_2]$ where $W_1 \cup W_2 = V_1$ are correct and store the non redundant minimal vertex cover for the subgraph G_{t_1} in W_1 and similarly, W_2 for G_{t_2}. Now we add $(w_1 + w_2 - w(V_1 \cap V_2), \alpha_1 + \alpha_2 - \alpha\{V_1 \cap V_2)))$ to $D[t, V_1, V_2]$.

What remains to be shown is that an undominated feasible solution $\mathcal{U}$ of Minimal Vertex Cover Knapsack in G is contained in $D[r, \{v\}, \emptyset]$. Let w be the weight of $\mathcal{U}$ and α be the value. Recall that $v \in \mathcal{U}$. For each t, we consider the subgraph G_t and observe how the minimal solution $\mathcal{S}'$ interacts with G_t. Let $\hat{V}_1, \hat{V}_2, \ldots, \hat{V}_m$ be components of $G_t \cap \mathcal{U}$ and let for each $1 \leqslant i \leqslant m$, $\mathcal{S}_i = X_t \cap \hat{V}_i$. Also, let $\hat{V}_0 = X_t \backslash \mathcal{U}$. Consider $\mathcal{S} = (\hat{V}_0, \hat{V}_1, \hat{V}_2, \ldots, \hat{V}_m)$. For each $\mathcal{S}_i$, we define subsets V_1 and V_2 such that $V_1, V_2 \subseteq \mathcal{S}_i$, and $V_1 \cup V_2 = \mathcal{S}_i$, $\forall i \in [m]$. The algorithm updates in $D[t, V_1, V_2]$ the pair (w', α') for the subsolution $(G_t\backslash$ edges incident on $V_2) \cap \mathcal{U}$. Therefore, $D[r, \{v\}, \emptyset]$ contains the pair (w, α). Thus, we are done.

Running Time. There are n choices for the fixed vertex v. Upon fixing v and adding it to each bag of $(\mathbb{T}, \mathcal{X})$, we consider the total possible number of states. Observe that the number of subproblems is small: for every node t, we have only $2^{|X_t|}$ choices for V_1 and V_2. Hence, the number of entries of the DP table is $\mathcal{O}\left(4^{\text{tw}} \cdot n\right)$. For each state, since we are keeping only undominated pairs, for each weight w there can be at most one pair with w as the first coordinate; similarly, for each value α there can be at most one pair with α as the second coordinate. Thus, the number of undominated pairs in each $D[t, V_1, V_2]$ is at most $\mathsf{min}\{s, p\}$ that can be maintained in time $\mathsf{min}\{s^2, p^2\}$. Updating the entries of the join nodes has the highest time complexity among all tree nodes, which is $\mathcal{O}\left(4^{\text{tw}} \cdot n^{\mathcal{O}(1)}\right)$. Hence, the overall running time of our algorithm is $\mathcal{O}\left(16^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \mathsf{min}\{s^2, p^2\}\right)$. ◀

We now design a fully FPT-time approximation scheme for the Minimal Vertex Cover Knapsack problem by rounding the values of the items so that $\alpha(\mathcal{V})$ is indeed a polynomial in n. The idea is to scale down the value of every vertex of the input instance so that the sum of values of the vertices that can be in the solution is polynomially bounded by input length and solve the scaled-down instance using the algorithm in Theorem 24. We usually scale down the values by dividing by $(\varepsilon \alpha_{\max})/n$. However, this approach does not work for our problems since $\alpha_{\max}$ is a lower bound on the optimal value for classical knapsack but not necessarily for our vertex cover knapsack variants. We tackle this issue by iteratively guessing upper and lower bounds of OPT thereby incurring an extra factor of poly$\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)$.

▶ **Theorem 25** (⋆). *For every $\varepsilon > 0$, there is an $(1 - \varepsilon)$-factor approximation algorithm for* Minimal Vertex Cover Knapsack *for optimizing the value of the solution and running in time $\mathcal{O}\left(16^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)\right)\right)$ where tw is the treewidth of the input graph.*

We obtain similar results for the other three variants of vertex cover knapsack for optimizing the value of the solution.

▶ **Corollary 26** (⋆). *For every $\varepsilon > 0$, there are $(1 - \varepsilon)$-factor approximation algorithms for* Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, *and* k-Vertex Cover Knapsack *for optimizing the value of the solution and running in time $\mathcal{O}\left(2^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)\right)\right)$.*

It turns out that we can use a similar idea as in Theorem 25 and Corollary 26 to design an FPT time $(1 + \varepsilon)$-approximation algorithm, parameterized by treewidth, for all the variants of vertex cover knapsack for minimizing the weight of the solution for every $\varepsilon > 0$.

▶ **Theorem 27** (⋆). *For every $\varepsilon > 0$, we have the following.*

1. *There is a $(1 + \varepsilon)$-factor approximation algorithm for* Minimal Vertex Cover Knapsack *for optimizing the weight of the solution and running in time $\mathcal{O}\left(16^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right)\right)\right)$ where tw is the treewidth of the input graph.*
2. *There are $(1 + \varepsilon)$-factor approximation algorithms for* Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, *and* k-Vertex Cover Knapsack *for optimizing the weight of the solution and running in time $\mathcal{O}\left(2^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right)\right)\right)$.*

6 Conclusion

We have studied the classical Knapsack problem with the graph theoretic constraints, namely vertex cover and its interesting variants like Minimum Vertex Cover Knapsack, Minimal Vertex Cover Knapsack, and k-Vertex Cover Knapsack. We further generalize this to hypergraphs and study Set Cover Knapsack and d-Hitting Set Knapsack. We have presented approximation algorithms for minimizing the size of the solution and proved that the approximation factors are the best possible that one hopes to achieve in polynomial time under standard complexity-theoretic assumptions. However, to maximize the value of the solution, we obtain strong inapproximability results. Fortunately, we show that there exist FPT algorithms parameterized by the treewidth of the input graph (for vertex cover variants of knapsack), which can achieve $(1-\varepsilon)$-approximate solution.

References

1 Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS J. Comput.*, 29(3):457–473, 2017. doi:10.1287/ijoc.2016.0742.

2 Flavia Bonomo and Diego de Estrada. On the thinness and proper thinness of a graph. *Discret. Appl. Math.*, 261:78–92, 2019. doi:10.1016/J.DAM.2018.03.072.

3 Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discret. Appl. Math.*, 196:62–71, 2015. doi:10.1016/J.DAM.2014.06.001.

4 Nicolas Boria, Federico Della Croce, and Vangelis Th Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/J.DAM.2014.06.001.

5 Stefano Coniglio, Fabio Furini, and Pablo San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *Eur. J. Oper. Res.*, 289(2):435–455, 2021. doi:10.1016/j.ejor.2020.07.023.

6 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

7 Peter Damaschke. Parameterized algorithms for double hypergraph dualization with rank limitation and maximum minimal vertex cover. *Discrete Optimization*, 8(1):18–24, 2011. doi:10.1016/J.DISOPT.2010.02.006.

8 Palash Dey, Ashlesha Hota, Sudeshna Kolay, and Sipra Singh. Knapsack with vertex cover, set cover, and hitting set, 2024. arXiv:arXiv:2406.01057.

9 Palash Dey, Sudeshna Kolay, and Sipra Singh. Knapsack: Connectedness, path, and shortest-path. In *Latin American Symposium on Theoretical Informatics*, pages 162–176. Springer, 2024. doi:10.1007/978-3-031-55601-2_11.

10 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

11 Herbert Fleischner, Gert Sabidussi, and Vladimir I. Sarvanov. Maximum independent sets in 3- and 4-regular hamiltonian graphs. *Discret. Math.*, 310(20):2742–2749, 2010. doi:10.1016/j.disc.2010.05.028.

12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

13 Steffen Goebbels, Frank Gurski, and Dominique Komander. The knapsack problem with special neighbor constraints. *Math. Methods Oper. Res.*, 95(1):1–34, 2022. doi:10.1007/s00186-021-00767-5.

14 Frank Gurski and Carolin Rehs. Solutions for the knapsack problem with conflict and forcing graphs of bounded clique-width. *Math. Methods Oper. Res.*, 89(3):411–432, 2019. doi:10.1007/s00186-019-00664-y.

15 Xin Han, Kazuo Iwama, Rolf Klein, and Andrzej Lingas. Approximating the maximum independent set and minimum vertex coloring on box graphs. In Ming-Yang Kao and Xiang-Yang Li, editors, *Algorithmic Aspects in Information and Management, Third International Conference, AAIM 2007, Portland, OR, USA, June 6-8, 2007, Proceedings*, volume 4508 of *Lecture Notes in Computer Science*, pages 337–345. Springer, 2007. doi:10.1007/978-3-540-72870-2_32.

16 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.40.

17 Stephan Held, William J. Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Program. Comput.*, 4(4):363–381, 2012. doi:10.1007/s12532-012-0042-3.

18 Mhand Hifi and Mustapha Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006. doi:10.1057/PALGRAVE.JORS.2602046.

19 Mhand Hifi and Mustapha Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research*, 34(9):2657–2673, 2007. doi:10.1016/J.COR.2005.10.004.

20 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/J.JCSS.2007.06.019.

21 Thiago Alcântara Luiz, Haroldo Gambini Santos, and Eduardo Uchoa. Cover by disjoint cliques cuts for the knapsack problem with conflicting items. *Oper. Res. Lett.*, 49(6):844–850, 2021. doi:10.1016/j.orl.2021.10.001.

22 Carlo Mannino, Gianpaolo Oriolo, Federico Ricci-Tersenghi, and L. Sunil Chandran. The stable set problem and the thinness of a graph. *Oper. Res. Lett.*, 35(1):1–9, 2007. doi:10.1016/J.ORL.2006.01.009.

23 Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *J. Graph Algorithms Appl.*, 13(2):233–249, 2009. doi:10.7155/jgaa.00186.

24 Ulrich Pferschy and Joachim Schauer. Approximation of knapsack problems with conflict and forcing graphs. *J. Comb. Optim.*, 33(4):1300–1323, 2017. doi:10.1007/s10878-016-0035-7.

25 Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

26 Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

27 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

28 Takeo Yamada, Seija Kataoka, and Kohtaro Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9), 2002.

Subsequence Matching and Analysis Problems for Formal Languages

Szilárd Zsolt Fazekas
Akita University, Japan

Tore Koß
University of Göttingen, Germany

Florin Manea
University of Göttingen, Germany

Robert Mercaş
Loughborough University, UK

Timo Specht
University of Göttingen, Germany

Abstract

In this paper, we study a series of algorithmic problems related to the subsequences occurring in the strings of a given language, under the assumption that this language is succinctly represented by a grammar generating it, or an automaton accepting it. In particular, we focus on the following problems: Given a string w and a language L, does there exist a word of L which has w as subsequence? Do all words of L have w as a subsequence? Given an integer k alongside L, does there exist a word of L which has all strings of length k, over the alphabet of L, as subsequences? Do all words of L have all strings of length k as subsequences? For the last two problems, efficient algorithms were already presented in [Adamson et al., ISAAC 2023] for the case when L is a regular language, and efficient solutions can be easily obtained for the first two problems. We extend that work as follows: we give sufficient conditions on the class of input-languages, under which these problems are decidable; we provide efficient algorithms for all these problems in the case when the input language is context-free; we show that all problems are undecidable for context-sensitive languages. Finally, we provide a series of initial results related to a class of languages that strictly includes the regular languages and is strictly included in the class of context-sensitive languages, but is incomparable to the of class context-free languages; these results deviate significantly from those reported for language-classes from the Chomsky hierarchy.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Stringology, String Combinatorics, Subsequence, Formal Languages, Context-Free Languages, Context-Sensitive Languages

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.28

Funding *Szilárd Zsolt Fazekas*: Supported by the JSPS Kakenhi Grant Number 23K10976.
Florin Manea: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) in the framework of the Heisenberg Programme project number 466789228.
Robert Mercaş: Supported by a Return Fellowship from the Alexander von Humboldt Foundation for a research stay at University of Göttingen, Germany.

1 Introduction

A string v is a subsequence of a string w, denoted $v \leq w$ in the following, if there exist (possibly empty) strings $x_1, \ldots, x_{\ell+1}$ and $v_1, \ldots, v_\ell$ such that $v = v_1 \cdots v_\ell$ and $w = x_1 v_1 \cdots x_\ell v_\ell x_{\ell+1}$. In other words, v can be obtained from w by removing some of its letters.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 28; pp. 28:1–28:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The concept of subsequence appears and plays important roles in many different areas of theoretical computer science. Prime examples are the areas of combinatorics on words, formal languages, automata theory, and logics, where subsequences are studied in connection to piecewise testable languages [71, 72, 37, 38], in connection to subword-order and downward-closures [31, 46, 45, 76, 77, 6]), in connection to binomial equivalence, binomial complexity, or to subword histories [66, 24, 49, 48, 69, 57, 67]. Subsequences are important objects of study also in the area of algorithm-design and complexity; to this end, we mention some classical algorithmic problems such as the computation of *longest common subsequences* or of the *shortest common supersequences* [16, 34, 36, 54, 56, 62, 8, 10], the testing of the Simon congruence of strings and the computation of the arch factorisation and universality of strings [32, 26, 73, 74, 18, 9, 19, 21, 28, 42, 22]; see also [44] for a survey on combinatorial pattern matching problems related to subsequences. Moreover, these algorithmic problems and other closely related ones have recently regained interest in the context of fine-grained complexity [12, 13, 3, 1, 2]. Nevertheless, subsequences appear also in more applied settings: for modelling concurrency [65, 70, 14], in database theory (especially *event stream processing* [7, 29, 78]), in data mining [50, 51], or in problems related to bioinformatics [11]. Interestingly, a new setting, motivated by database theory [39, 40, 25], considers subsequences of strings, where the substrings occurring between the positions where the letters of the subsequence are embedded are constrained by regular or length constraints; a series of algorithmic results (both for upper and lower bounds) on matching and analysis problems for the sets of such subsequences occurring in a string were obtained [20, 41, 5, 55].

The focus of this paper is the study of the subsequences of strings of a formal language, the main idea behind it being to extend the fundamental problems related to matching subsequences in a string and to the analysis of the sets of subsequences of a single string to the case of sets of strings. To this end, grammars (or automata) are succinct representations of (finite or infinite) sets of strings they generate (respectively, accept), so we are interested in matching and analysis problems related to the set of subsequences of the strings of a language, given by the grammar generating it (respectively, the automaton accepting it). This research direction is, clearly, not new. To begin with, we recall the famous result of Higman [33] which states that the downward closure of every language (i.e., the set of all subsequences of the strings of the respective language) is regular. Clearly, it is not always possible to compute an automaton accepting the downward closure of a given language, but gaining a better understanding when it is computable is an important area of research, as the set of subsequences of a language plays meaningful roles in practical applications (e.g., abstractions of complex systems, see [76, 77, 6] and the references therein). Computing the downward closure of a language is a general (although, often inefficient) way to solve subsequence-matching problems for languages; for instance, we can immediately check, using a finite automaton for the downward closure, if a string occurs as subsequence of a string of the respective language. However, it is often the case that more complex analysis problems regarding the subsequences occurring in the strings of a language cannot be solved efficiently (or, sometimes, at all) using the downward closure; such a problem is to check if a given string occurs as subsequence in all the strings of a language (chosen from a complex enough class, such as the class of context-free languages).

As a direct predecessor of this paper, motivated by similar questions, [4] approached algorithmic matching and analysis problems related to the universality of regular languages (for short, REG). More precisely, a string over Σ is called k-universal if its set of subsequences includes all strings of length k over Σ; the study of these universal strings was the focus of many recent works [9, 19, 68] and the motivation for studying universality properties in the context of subsequences is discussed in detail in, e.g., [19, 4]. The main problems addressed in [4] are the following: for $L \in \text{REG}$, over the alphabet Σ, and a number k, decide if there exists a k-universal string in L (respectively, if all strings of L are k-universal). The authors of [4] discussed efficient algorithms solving these problems and complexity lower bounds. In this paper, we extend the work of [4] firstly by proposing a more structured approach for the algorithmic study of the subsequences occurring in strings of formal languages and secondly by considering more general classes of languages, both from the Chomsky hierarchy (such as the class of context-free languages or that of context-sensitive languages) and non-classical (the class of languages accepted by deterministic finite automata with translucent letters).

Our work on subsequence-matching and analysis problems in languages defined by context-free grammars (for short, CFG) also extends a series of results related to matching subsequences in strings given as a straight line program (for short, SLP; a CFG generating a single string), or checking whether a string given as an SLP is k-universal, for some given k, e.g., see [52, 68]. In our paper, we consider the case when the input context-free languages and the CFGs generating them are unrestricted.

The approached problems and an overview of our results. As mentioned above, we propose a more structured approach for matching- and analysis-problems related to subsequences of the strings of a formal language. More precisely, we define and investigate the following five problems.

▶ **Problem 1** ($\exists$-Subsequence)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and a string w, is there a string $v \in L$ such that $w \leq v$?*

▶ **Problem 2** ($\forall$-Subsequence)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and a string w, do we have for all strings $v \in L$ that $w \leq v$?*

▶ **Problem 3** ($\exists$-k-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and integer k, check if there is a k-universal string in L.*

▶ **Problem 4** ($\forall$-k-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating) it and integer k, check if all strings of $L(M)$ are k-universal.*

Alternatively, strictly from the point of view of designing an algorithmic solution, the problem above can be approached via its complement: that is, deciding if there exists at least one string in $L(M)$ which is not k-universal.

▶ **Problem 5** (∞-universal)**.** *Given a language L by a machine (grammar) M accepting (respectively, generating it) decide if there exist m-universal strings in L, for all positive integers m.*

To give some intuition on our terminology, Problems 1 and 3 can be seen as *matching problems* (find a string which contains a certain subsequence or set of subsequences), while the other three problems are *analysis problems* (decide properties concerning multiple strings of the language).

Going a bit more into details, in the main part of this paper, we investigate these problems for the case when the language L is chosen from the class of context-free languages (for short, CFL; given by CFGs in Chomsky normal form), or from the class of context-sensitive languages (for short, CSL; given by context-sensitive grammars), or from the class of languages accepted by deterministic finite automata with translucent letters (given by an automaton of the respective kind). The choice of presentation of the languages from given classes, unsurprisingly, makes a big difference w.r.t. hardness. For instance, certain singleton languages can be encoded by SLPs (essentially CFGs) exponentially more succinctly than by classical DFA, which of course introduces significant extra computation into solving subsequence-related queries [52, 68]. But, before approaching these classes of languages, we provide a series of general decidability results on these five problems, for which the choice of grammar or automaton as the way of specifying the input language L is not consequential.

For short, our results are the following. We first give (in Section 3) a series of simple sufficient conditions on a class $\mathcal{C}$ of languages (related to the computation of downward closures as well as to decidability properties for the respective class) which immediately lead to decision procedures for the considered problems; however, these procedures are inherently inefficient, even for classes such as CFL. In this context, generalizing the work of [4], we approach (in subsequent sections of this paper) each of the above problems for $\mathcal{C}$ being the class CFL and, respectively, the class CSL. While all the problems are undecidable for CSLs, we present efficient algorithms for the case of CFLs. In particular, the results obtained for CFL are similar to the corresponding results obtained for REG (i.e., if a problem was solvable in polynomial or FPT-time for REG, we obtain an algorithm from the same class for CFL). In that regard, it seemed natural to search for a class of languages which does not exhibit this behaviour, while retaining the decidability of (at least some of) these problems. To this end, we identify the class of languages accepted by deterministic finite automata with translucent letters (a class of automata which does not process the input in a sequential fashion) and show (in the final section of this paper) a series of initial promising results related to them.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ denote the natural numbers and set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ as well as $[n] = \{1, \ldots, n\}$ and $[i, n] = \{i, i+1, \ldots, n\}$, for all $i, n \in \mathbb{N}_0$ with $i \leq n$.

An *alphabet* $\Sigma = \{1, 2, \ldots, \sigma\}$ is a finite set of symbols, called *letters*. A *string* w is a finite concatenation of letters from a given alphabet with the number of these letters giving its *length* $|w|$. The string with no letters is the *empty string* ε of length 0. The set of all finite strings over the alphabet Σ, denoted by Σ^*, is the free monoid generated by Σ with concatenation as operation. A subset $L \in \Sigma^*$ is called a *(formal) language*. Let Σ^n denote all strings in Σ^* exactly of length $n \in \mathbb{N}_0$.

For $1 \leq i \leq j \leq |w|$ denote the i^{th} letter of w by $w[i]$ and the *factor* of w starting at position i and ending at position j as $w[i, j] = w[i] \cdots w[j]$. If $i = 1$ the factor is also called a prefix, while if $j = |w|$ it is called a suffix of w. For each $a \in \Sigma$ set $|w|_a = |\{i \in [|w|] \mid w[i] = a\}|$.

Let $\mathrm{alph}(w)$ denote the set of all letters of Σ occurring in w. A length n string $u \in \Sigma^*$ is called *subsequence* of w, denoted $u \leq w$, if $w = w_1 u[1] w_2 u[2] \cdots w_n u[n] w_{n+1}$, for some $w_1, \ldots, w_{n+1} \in \Sigma^*$. For $k \in \mathbb{N}_0$, a string $w \in \Sigma^*$ is called *k-universal* (w.r.t. Σ) if every $u \in \Sigma^k$ is a subsequence of w. The *universality-index* $\iota(w)$ is the largest k such that w is k-universal.

▶ **Definition 6.** *The arch factorization of a string $w \in \Sigma^*$ is given by $w = ar_1(w) \cdots ar_{\iota(w)}(w) r(w)$ with $\iota(ar_i(w)) = 1$ and $ar_i(w)[|ar_i(w)|] \notin \mathrm{alph}(ar_i(w)[1, |ar_i(w)| - 1])$, for all $i \in [1, \iota(w)]$. Furthermore, $\mathrm{alph}(r(w)) \subsetneq \Sigma$ applies. The strings $ar_i(w)$ are called* arches *and $r(w)$ is called* the rest *of w.*
The modus $m(w)$ of w is defined as the concatenation of the last letters of each arch: $m(w) = ar_1(w)[|ar_1(w)|] \cdots ar_{\iota(w)}(w)[|ar_{\iota(w)}(w)|]$.

As an example, in the arch factorisation $w = (bca)\cdot(accab)\cdot(cab)\cdot b$ of $w \in \{a, b, c\}^*$, the parentheses denote the three arches and the rest $r(w) = b$. Further, we have $\iota(w) = 3$ and $m(bcaaccabcabb) = abb$. For more details about the arch factorization and the universality index see [32, 9].

A string v is an absent subsequence of another string w if v is not a subsequence of w [42, 43]. A shortest absent subsequence of a string w (for short, $\mathrm{SAS}(w)$) is an absent subsequence of w of minimal length, i.e., all subsequences of shorter length are found in w. We note that, for a given word w and some letter $a \notin \mathrm{alph}(r(w))$, an SAS of w is $m(w)a$ [32, 42]. An immediate observation is that any string v which is an $\mathrm{SAS}(w)$ satisfies $|v| = \iota(w) + 1$.

In this paper, we work with absent subsequences of a word w which are the shortest among all absent subsequences of w and, additionally, start with a and end with b, for some $a \in \Sigma \cup \{\varepsilon\}$ and $b \in \Sigma$. Such a shortest string which starts with a and ends with b and is an absent subsequence of w is denoted $\mathrm{SAS}_{a,b}(w)$. For instance, an $\mathrm{SAS}_{\varepsilon,b}(w)$, for some $b \notin \mathrm{alph}(r(w))$, is an $\mathrm{SAS}(w)$, such that its starting letter is not fixed, but the ending one must be b.

▶ **Definition 7.** *A grammar over an alphabet Σ is a 4-tuple $G = (V, \Sigma, P, S)$ consisting of: a set $V = \{A, B, C, \dots\}$ of non-terminal symbols, a set $\Sigma = \{a, b, c, \dots\}$ of terminal symbols with $V \cap \Sigma = \emptyset$, a non-empty set $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ of productions and a start symbol $S \in V$.*

We represent productions $(p, q) \in P$ by $p \to q$. In G, $u = xpz$ with $x, z \in (V \cup \Sigma)^*$ is directly derivable to $v = xqz$ if a production $(p, q) \in P$ exists; in this case, we write $u \Rightarrow_G v$; the subscript G is omitted when no confusion arises. More generally, for $m \in \mathbb{N}$, we say that u is derivable to v in m steps (denoted $w \Rightarrow_G^m v$) if there exist strings $w_0, w_1, \dots, w_m \in (V \cup \Sigma)^*$ (called sentential forms) with $u = w_0 \Rightarrow_G w_1 \wedge w_1 \Rightarrow_G w_2 \wedge \cdots \wedge w_{m-1} \Rightarrow_G w_m = v$. If u is derivable to v in m steps, for some $m \in \mathbb{N}_0$, we write $u \Rightarrow_G^* v$, i.e., $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$. With $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ we denote the language generated by G. We call a derivation a sequence $S \Rightarrow \cdots \Rightarrow w \in L(G)$. The number of steps used in the derivation is the derivation's length.

▶ **Definition 8.** *A grammar $G = (V, \Sigma, P, S)$ with $P \subseteq V \times (V \cup \Sigma)^+$ is a* context-free *grammar (for short, CFG). A language L is a context-free language (for short, CFL) if and only if there is a CFG G with $L(G) = L$.*
A grammar $G = (V, \Sigma, P, S)$, where for all $(p, q) \in P$ we have $|p| \leq |q|$, is a context-sensitive *grammar (for short, CSG). A language L is a context-sensitive language (for short, CSL) if and only if there is a CSG G with $L(G) = L$.*

The definitions above tacitly assume that CFLs and CSLs do not contain the empty string ε. Indeed, for the problems considered here, we can make this assumption. Whether $\varepsilon \in L$ or not plays no role in deciding Problems 1, 3, and 5, while $\varepsilon \in L$ immediately leads to a negative answer for Problem 2 (unless $w = \varepsilon$) and Problem 4 (unless $k = 0$). So, for simplicity, we only address languages that, by definition, *do not* contain the empty string (see also the discussions in [47, 35] about how the presence of ε in formal languages can be handled).

Also, note that every unary CFL is regular [64], so when discussing our problems for the class of CFLs we assume that the input languages are over an alphabet with at least two letters.

▶ **Definition 9.** *A CFG $G = (V, \Sigma, P, S)$ is in Chomsky normal form (CNF) if and only if $P \subseteq V \times (V^2 \cup \Sigma)$ and, for all $A \in V$, there exist some $w_A, w'_A, w''_A \in \Sigma^*$ such that $A \Rightarrow^* w_A$ and $S \Rightarrow^* w'_A A w''_A$ (these last two properties essentially say that every non-terminal of G is* useful*).*

When we discuss our problems in the case of CFLs, we assume our input is a CFG G in CNF. This does not change our results since, according to [47] and the references therein, we can transform any grammar G in polynomial time into a CFG G' in CNF such that $|G'| \in \mathcal{O}(|G|^2)$ and $L(G) = L(G')$, where $|G|$ refers to the size of a grammar determined in terms of total size of its productions.

In some cases it may be easier to view derivations in a CFG G as a derivation (parse) tree. These are rooted, ordered trees. The inner nodes of such trees are labeled with non-terminals and the leaf-nodes are labeled with symbols $X \in (V \cup \Sigma)$. An inner node A has, from left to right, the children $X_1, \ldots, X_k$, for some integer $k \geq 1$, if the grammar contains the production $A \to X_1 \cdots X_k$. As such, if we concatenate, from left to right, the leaves of a derivation tree T with root A we get a string α (called in the following the border of T) such that $A \to^* \alpha$. The depth of a derivation tree is the length of the longest simple-path starting with the root and ending with a leaf (i.e., the number of edges on this path). If G is in CNF, then all its derivation trees are binary.

▶ **Definition 10.** *For any language $L \subset \Sigma^*$ the* downward closure *$L{\downarrow}$ of L is defined as the language containing all subsequences of strings of L, i.e., $L{\downarrow} = \{v \in \Sigma^* \mid \exists w \in L : v \leq w\}$. The complementary notion of the* upward closure *$L{\uparrow}$ of a language L is the language containing all supersequences of strings in L, i.e., $L{\uparrow} = \{w \in \Sigma^* \mid \exists v \in L : v \leq w\}$.*

Our problems focus on properties of formal languages, and Problems 3, 4 and 5 are strongly connected to universality seen as a property of a language, therefore we extend the concept of universality to formal languages. We distinguish between two different ways of analyzing the universality of a language.

▶ **Definition 11.** *Let $L \subseteq \Sigma^*$ be a language. We call L k-universal universal if for every $w \in L$ it holds that $\iota(w) \geq k$. The universal universality index $\iota_\forall(L)$ is the largest k, such that L is k-universal universal. We call L k-existential universal if a string $w \in L$ exists with $\iota(w) \geq k$. The existential universality index $\iota_\exists(L)$ is the largest k, such that L is k-existential universal. In all the definitions above, the universality index of words and, respectively, languages is computed w.r.t. Σ.*

In case of a singleton language $L = \{w\}$ it holds that $\iota_\exists(L) = \iota_\forall(L) = \iota(w)$. In general the universal universality index $\iota_\forall(L)$ is the infimum of the set of all universality indices of strings in L and therefore is lower bounded by 0 and upper bounded by $\iota(w)$, for any $w \in L$ (so it is finite, for $L \neq \emptyset$). The existential universality index $\iota_\exists(L)$ is the supremum of the set of all universality indices of strings in L and, as such, can be infinite. In this setting the answer to Problem 3 and, respectively, Problem 4, can be solved by computing $\iota_\exists(L)$ and, respectively, $\iota_\forall(L)$, and then checking whether $k \leq \iota_\exists(L)$ and, respectively, $k \leq \iota_\forall(L)$. Furthermore, Problem 5 asks whether $\iota_\exists(L)$ is infinite or not. The following two lemmas are not hard to show.

▶ **Lemma 12.** *Given a string $w \in \Sigma^*$, with $|w| = n$ and $|\Sigma| = \sigma$, we can construct in time $\mathcal{O}(n\sigma)$ a minimal DFA, with $n + 1$ states, accepting the set of strings which have w as a subsequence.*

▶ **Lemma 13.** *For $k > 0$ and an alphabet Σ with $|\Sigma| = \sigma$ we can construct in time $\mathcal{O}(2^{\sigma} k \operatorname{poly}(\sigma))$ a minimal DFA, with $(2^{\sigma} - 1)k + 1$ states, accepting the set of k-universal strings over Σ.*

The computational model we use to state our algorithms is the standard unit-cost word RAM with logarithmic word-size ω (meaning that each memory word can hold ω bits). It is assumed that this model allows processing inputs of size n, where $\omega \geq \log n$; in other words, the size n of the data never exceeds (but, in the worst case, is equal to) 2^{ω}. Intuitively, the size of the memory word is determined by the processor, and larger inputs require a stronger processor (which can, of course, deal with much smaller inputs as well). Indirect addressing and basic arithmetical operations on such memory words are assumed to work in constant time. Note that numbers with ℓ bits are represented in $O(\ell/\omega)$ memory words, and working with them takes time proportional to the number of memory words on which they are represented. This is a standard computational model for the analysis of algorithms, defined in [23].

Our algorithms have languages as input, that is sets of strings over some finite alphabet. Therefore, we follow a standard stringology-assumption, namely that we work with an *integer alphabet*: we assume that this alphabet is $\Sigma = \{1, 2, \ldots, \sigma\}$, with $|\Sigma| = \sigma$, such that σ fits in one memory word. For a more detailed general discussion on the integer alphabet model see, e. g., [17]. In all problems discussed here, the input language is given as a grammar generating it or as an automaton accepting it. We assume that all the sets defining these generating/accepting devices (e.g., set of non-terminals, set of states, set of final states, relation defining the transition function or derivation, etc.) have at most 2^{ω} elements and their elements are integers smaller or equal to 2^{ω} (i.e., their cardinality and elements can be represented as integers fitting in one memory word). In some of the problems discussed in this paper, we assume that we are given a number k. Again, we assume that this integer fits in one memory word.

One of our algorithms (for Problem 3 in the case of CFL, stated in Theorem 23) runs in exponential time and uses exponential space w.r.t. the size of the input. In particular, both the space and time complexities of the respective algorithm are exponential, with constant base, in σ (the size of the input alphabet) but polynomial w.r.t. all the other components of the input. To avoid clutter, we assume that our exponential-time and -space algorithm runs on a RAM model where we can allocate as much memory as our algorithms needs (i.e., the size of the memory-word ω is big enough to allow addressing all the memory we need in this algorithm in constant time); for the case of $\sigma \in O(1)$, this additional assumption becomes superfluous, and for non-constant σ we stress out that the big size of memory words is only used for building large data structures, but not for speeding up our algorithms by, e.g., allowing constant-time operations on big numbers (that is, numbers represented on more than $c \log N$ bits, for some constant c and N being the size of the input).

3 General Results

We consider the problems introduced in Section 1, for the case when the language L is chosen from a class $\mathcal{C}$, and give a series of sufficient conditions for them to be decidable.

Consider a class $\mathcal{G}$ of grammars (respectively, a class $\mathcal{A}$ of automata) generating (respectively, accepting) the languages of the class $\mathcal{C}$. For simplicity, for the rest of this section, we assume that in all the problems we take as input a grammar G_L such that $L(G_L) = L$, but note that all the results hold for the case when we consider that the languages are given by an automaton from the class $\mathcal{A}$ accepting them.

Let $\mathcal{C}'$ be the class of languages $L \cap R$, where $L \in \mathcal{C}$ and $R \in \text{REG}$. We use two hypotheses:

H1. Given a grammar G of the class $\mathcal{G}$ we can algorithmically construct a non-deterministic finite automaton A accepting the downward closure of $L(G)$.

H2. Given a grammar G of the class $\mathcal{G}$ and a non-deterministic finite automaton A, we can algorithmically decide whether the language $L(G) \cap L(A)$ is empty.

First we show that, under H1, Problems 1, 3, and 5 are decidable.

▶ **Theorem 14.** *If H1 holds, then Problems 1, 3, and 5 are decidable.*

Proof. We start by observing that the following straightforward properties hold:

- for a string w, there exists $v \in L$ such that $w \leq v$ if and only if $w \in L\!\downarrow$.
- for an integer $k > 0$, there exists $v \in L$ such that v is k-universal if and only if there exists $v' \in L\!\downarrow$ such that v' is k-universal.

In each of Problems 1, 3, and 5, we are given a grammar G generating the language L. According to H1, we construct a non-deterministic automaton A accepting $L\!\downarrow$, the downward closure of L.

For Problem 1, it is sufficient to check if $L(A) = L\!\downarrow$ contains the string w, which is clearly decidable. For Problem 3 we need to decide if L contains a k-universal string. By our observations, it is enough to check if $L\!\downarrow$ contains a k-universal string. This can be decided, for the automaton A, according to [4]. For Problem 5 we need to decide if L contains a k-universal string, for all $k \leq 0$. This is also decidable, for A, according to the results of [4]. ◀

Secondly, we show that, under H2, Problems 1, 2, 3, and 4 are decidable.

▶ **Theorem 15.** *If H2 holds, then Problems 1, 2, 3, and 4 are decidable.*

Proof. In all the inputs of Problems 1, 2, 3, and 4 when considering CFL and CSL, we are given a grammar G, which generates the language L.

For Problems 1 and 2, by Lemma 12 we construct a DFA B accepting the regular language $w\!\uparrow$ of strings which have w as a subsequence. If the intersection of L (given as the grammar G which generates it) and $L(B)$ is empty, which is decidable, under H2, then the considered instance of Problem 1 is answered negatively; otherwise, it is answered positively. By making the final state of B non-final, and all the other states final, we obtain a DFA B' which accepts $\Sigma^* \setminus w\!\uparrow$. If the intersection of L and $L(B')$ is empty, then the answer to the considered instance of Problem 2 is positive; otherwise, it is negative.

For Problems 3 and 4, by Lemma 13 we construct a DFA B accepting the regular language of k-universal strings. If the intersection of L and $L(B)$ is empty, then the answer to the considered instance of Problem 3 is negative; otherwise, it is positive. By making the final state of B non-final, and all the other states final, we obtain a DFA B' which accepts exactly all strings which are not k-universal. If the intersection of L and $L(B')$ is empty, then the answer to the considered instance of Problem 4 is positive; otherwise, it is negative. ◀

It is worth noting that, even for classes which fulfill both hypotheses above (such as the CFLs [75, 35]), there are several reasons why the algorithms resulting from the above theorems are not efficient. On the one hand, constructing an automaton which accepts the downward closure of a language is not always possible, and even when this construction is possible (when the language is from a class for which H1 holds) it cannot always be done efficiently. For instance, in the case of CFLs, this may take inherently exponential time w.r.t. the size of the input grammar [30]; in this paper, we present more efficient algorithms for

Problems 1, 2, 3, and 4 in the case of CFLs, which do not rely on Theorem 15. On the other hand, the results of Theorem 15 rely, at least partly, on the construction of a DFA accepting all k-universal strings, which takes exponential time in the worst case, as it may have an exponential number of states (both w.r.t. the size of the input alphabet and w.r.t. the binary representation of the number k, which is given as input for some of these problems).

Interestingly, the class CSL does not fulfil any of the above hypotheses. In fact, as our last general result, we show that all five problems we consider here are undecidable for CSL.

▶ **Theorem 16.** *Problems 1, 2, 3, 4, 5 are undecidable for the class of CSL, given as CSGs.*

Proof. To obtain the undecidability of all the problems, we show reductions from the emptiness problem for Context Sensitive Languages. Assume that we have a CSL L, specified by a grammar G, as the input for the emptiness problem for CSL. Assume L is over the alphabet $\Sigma = \{1, \ldots, \sigma\}$, and that the CSG G, has the starting symbol S. Let 0 be a fresh letter (i.e., $0 \notin \Sigma$).

To show the undecidability of Problems 1 and 2, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the productions $S' \to \sigma S$ and $S' \to 0$.

It is immediate that there exists a string $w \in L(G')$ which contains σ as a subsequence, if and only if $L(G)$ is not empty. Furthermore, all strings of $L(G')$ contain 0 as a subsequence (that is, the production $S' \to \sigma S$ is not the first production in the derivation of any terminal string) if and only if $L(G)$ is empty. As the emptyness problem is undecidable for CSL (given as grammars), it follows that Problems 1 and 2 are also undecidable for this class of languages.

To show the undecidability of Problem 3, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the production $S' \to (12 \cdots \sigma)S$. Clearly, $L(G')$ contains a 1-universal string (over Σ) if and only if $L(G) \neq \emptyset$. Thus, it follows that Problem 3 is also undecidable for this class of languages.

To show the undecidability of Problem 4, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and the productions $S' \to 012 \cdots \sigma$ and $S' \to S$. Clearly, all the strings of $L(G')$ are 1-universal (over $\Sigma \cup \{0\}$) if and only if $L(G) = \emptyset$ (as any string which would be derived in G' starting with the production $S' \to S$ would not contain 0). Hence, Problem 4 is also undecidable for CSL.

To show the undecidability of Problem 5, we construct a new grammar G' which has all the non-terminals, terminals, and productions of G and, additionally, G' has a new starting symbol S' and a fresh non-terminal R and the productions $S' \to 012 \cdots \sigma$, $S' \to RS$, $R \to 01 \cdots \sigma R$, and $R \to 01 \cdots \sigma$. Clearly, $L(G')$ contains m-universal strings (over $\Sigma \cup \{0\}$), for all $m \geq 1$, if and only if $L(G) \neq \emptyset$ (as we can use R to pump arches in the strings of $L(G')$ if and only if there exists at least one derivation where S can be derived to a terminal string). Accordingly, Problem 5 is also undecidable for CSL. ◀

Given that all the problems become undecidable for $\mathcal{C} = \text{CSL}$, we now focus our investigation on classes of languages strictly contained in the class of CSLs.

4 Problems 1 and 2

For the rest of this section, assume that $|w| = m$ and $|\Sigma| = \sigma$. Let us begin by noting that Problems 1 and 2 can be solved in polynomial time for the class REG following the approach of Theorem 15. Indeed, in this case, we assume that L is specified by the NFA A, with s

states, with $L(A) = L$, and then we either have to check the emptiness of the intersection of $L = L(A)$ with the language accepted by the deterministic automaton constructed in Lemma 12, or, respectively, with the complement of this language; both these tasks clearly take polynomial time.

We now consider the two problems for the class of CFLs. We first recall the following folklore lemma (see, e.g., [35]).

▶ **Lemma 17.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF, and let $A = (Q, \Sigma, q_0, F, \delta)$ be a DFA. Then we can construct in polynomial time a CFG G_A in CNF such that $L(G_A) = L(G) \cap L(A)$.*

We can now state the main result of this section. We can apply Lemma 17, for Problem 1, to the input CFG and the DFA constructed in Lemma 12, or, for Problem 2, to the input CFG and the complement of the respective DFA. In both cases, we compute a CFG in CNF generating the intersection of a CFL and a REG language, and we have to check whether the language generated by that grammar is empty or not; all these can be implemented in polynomial time.

▶ **Theorem 18.** *Problems 1 and 2, for an input grammar with n non-terminals and an input string of length m, are decidable in polynomial time for CFL.*

5 Problems 3 and 5

Let us begin by noting that in [4] it was shown that for a given NFA A with s states (with input alphabet Σ, where $|\Sigma| = \sigma$) and an integer $k \geq 0$, we can decide whether $L(A)$ contains a k-universal string (i.e., Problem 3 for the class REG) in time $\mathcal{O}(\text{poly}(s, \sigma)2^{\sigma})$; in other words, Problem 3 is fixed parameter tractable (FPT) w.r.t. the parameter σ. A polynomial time algorithm (running in $\mathcal{O}(\text{poly}(s, \sigma)$ time) was given for Problem 5, relying on the observation that, given an NFA A, the language $L(A)$ contains strings with arbitrarily large universality if and only if A contains a state q, which is reachable from the initial state and from which one can reach a final state, and a cycle which contains this state, whose label is 1-universal. Coming back to Problem 3 for REG, the same paper shows that it is actually NP-complete. This is proved by a reduction from the Hamiltonian Path problem (HPP, for short), in which a graph with n vertices, the input of HPP, is mapped to an input of Problem 3 consisting in an automaton with $\mathcal{O}(n^2)$ states over an alphabet of size n. This reduction also implies that, assuming that the Exponential Time Hypothesis (ETH, for short) holds, there is no $2^{o(\sigma)}\text{poly}(s, \sigma)$ time algorithm solving Problem 3 (as this would imply the existence of an $2^{o(n)}$ time algorithm solving HPP); see [53] for more details related to the ETH and HPP.

Further, we consider Problems 3 and 5 for the class CFL, and we assume that, in both cases, we are given a CFL L by a CFG $G = (V, \Sigma, P, S)$ in CNF, with n non-terminals, over an alphabet Σ, with σ letters, and an integer $k \geq 1$ (in binary representation).

To transfer the lower bound derived for Problem 3 in the case of REG (specified as NFAs) to the larger class of CFLs, we recall the folklore result that a CFG in CNF can be constructed in polynomial time from an NFA (by constructing a regular grammar from the NFA, and then putting the grammar in CNF, see [35]). So, the same reduction from [4] can be used to show that, assuming ETH holds, there is no $2^{o(\sigma)}\text{poly}(n, \sigma)$ time algorithm solving Problem 3. This reduction shows also that Problem 3 is NP-hard; whether this problem is in NP remains open.

We now focus on the design of a $2^{\mathcal{O}(\sigma)}\text{poly}(n,\sigma)$ time algorithm solving Problem 3 (which would also show that this problem is FPT) and show that Problem 5 can be solved in polynomial time. Let us recall that Problem 5 requires deciding whether $\iota_\exists(L)$ is finite, and, if yes, Problem 3 requires checking whether $\iota_\exists(L) \geq k$.

We start by introducing a new concept which leads to a series of combinatorial observations.

▶ **Definition 19.** *Let $G = (V, \Sigma, P, S)$ be a CFG. A non-terminal $A \in V$ generates a* 1-universal cycle *if and only if there exists a derivation $A \Rightarrow^* w_1 A w_2$ of the grammar G with $w_1, w_2 \in \Sigma^*$ and $\max(\iota(w_1), \iota(w_2)) \geq 1$.*

We can show the following result.

▶ **Lemma 20.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF and $L = L(G)$. Then $\iota_\exists(L)$ is infinite if and only if there exists a non-terminal $X \in V$ such that X generates a 1-universal cycle.*

Proof. Assume we have a non-terminal $A \in V$ which generates a *1-universal cycle*. This means that there exists a derivation $A \Rightarrow^* w_1 A w_2$ with $w_1, w_2 \in \Sigma^*$ and $\iota(w_1) \geq 1$ or $\iota(w_2) \geq 1$. As G is in CNF, we have that there exist $w'_A, w''_A \in \Sigma^*$ and the derivation $S \Rightarrow^* w'_A A w''_A$, and, also, that there exists $w_A \in \Sigma^*$ such that $A \Rightarrow^* w_A$. We immediately get that, for all $n \geq 1$, the following derivation is valid: $S \Rightarrow^* w'_A A w''_A \Rightarrow^* w'_A w_1 A w_2 w''_A \Rightarrow^* w'_A (w_1)^2 A (w_2)^2 w''_A \Rightarrow^* w'_A (w_1)^n A (w_2)^n w''_A \Rightarrow^* w'_A (w_1)^n w_A (w_2)^n w''_A = w$. As $\iota(w_1) \geq 1$ or $\iota(w_2) \geq 1$, it follows that $\iota(w) \geq n$. So, $\iota_\exists(L)$ is infinite.

We now show the converse implication. More precisely, we show by induction on the number of non-terminals of G that if $\iota_\exists(L(G))$ is infinite then G has at least one useful non-terminal $X \in V$ such that X has a 1-universal cycle. For this induction proof, we can relax the restrictions on G: more precisely, we still assume that the set P of productions of G fulfils $P \subseteq V \times (V^2 \cup \Sigma)$ but do not require that every non-terminal of G is useful; it suffices to require the starting symbol to be useful.

The result is immediate if G has a single non-terminal, i.e., the start symbol S. We now assume that our statement holds for CFLs generated by grammars with at most m non-terminals, and assume that L is a CFL generated by a CFG G with $m+1$ non-terminals. We want to show that G has at least one useful non-terminal $X \in V$ such that X has a 1-universal cycle. We can assume, w.l.o.g., that S does not have a 1-universal cycle (otherwise, the result already holds).

Now, consider for each useful $A \in V \setminus \{S\}$ the CFG (which fulfills the requirements of our statement) $G_A = (V \setminus \{S\}, \Sigma, A, P')$, where P' is obtained from P by removing all productions involving S. Clearly, if there exists some $A \in V$ such that $\iota_\exists(L(G_A))$ is infinite, then, by induction, G_A contains a useful non-terminal $X \in V$ such that X has a 1-universal cycle. As G_A is obtained from G by removing some productions and one non-terminal, it is clear that X also has a 1-universal cycle in G and is also useful in G, so our statement holds. Let us now assume, for the sake of a contradiction, that, for each useful $A \in V$, there exists an integer $N_A \geq 1$ such that $\iota_\exists(L(G_A)) \leq N_A$. Take $N = 1 + \max\{N_A \mid A \in V\}$. As $\iota_\exists(L)$ is infinite, there exists a string $w \in L(G)$ with $\iota(w) \geq 2N+3$. Since $w \in L(G)$, $S \Rightarrow^* w$ holds.

Let T_S be the derivation tree of w with root S and note that all non-terminals occurring in T_S are useful. Let p the longest simple path of T_S starting in S and having the end-node S (in the case when there are more such paths, we simply choose one of them). We denote by T_S^p the sub-tree of T_S rooted in the end-node of p. If w' is the string obtained by reading the leaves of T_S^p left-to-right, then we have the following derivation corresponding to T_S: $S \Rightarrow v_S S v'_S \Rightarrow^* v_S w' v'_S = w$, where $v_S, v'_S \in \Sigma^*$. Since, by our assumption, S does not have a 1-universal cycle, we get that $\iota(v_S) = 0$, $\iota(v'_S) = 0$, and that $\iota(w') \geq 2N+1$.

Further, we consider T_S^p, and note that no other node of this tree, except the root, is labelled with S. Assume that the first step in the derivation $S \Rightarrow^* w'$ is $S \Rightarrow AB$, for some non-terminals $A, B \in V$ and production $S \to AB$, and that the children of the root S in the tree T_S^p are the sub-trees T_A and T_B. Let w_A be the border of T_A and w_B be the border of T_B. Clearly, it follows that at least one of the strings w_A and w_B is N-universal. We can assume, w.l.o.g., that $\iota(w_A) \geq N$. But $w_A \in L(G_A)$ and $\iota_\exists(L(G_A)) < N$ (by the definition of N). This is a contradiction with our assumption that $\iota(L(G_X))$ is finite, for all $X \in V$. So, there exists $X \in V$ for which $\iota_\exists(L(G_X))$ is infinite and, as we have seen, this means that our statement holds. ◀

So, according to Lemma 20, if the CFG G, which is the input of our problem, contains at least one non-terminal $X \in V$ which has a 1-universal cycle, we answer positively the instances of Problems 3 and 5 defined by G and, in the case of Problem 3, additionally by an integer $k \geq 1$. Next, we show that one can decide in polynomial time whether such a non-terminal exists in a grammar. However, if G does not contain any non-terminal with a 1-universal cycle, while the instance of Problem 5 is already answered negatively, it is unclear how to answer Problem 3. To address this, we try to find a way to efficiently construct a string of maximal universality index, and, for that, we need another combinatorial result.

▶ **Lemma 21.** *Let $G = (V, \Sigma, P, S)$ be a CFG in CNF, with $|V| = n$, $|\Sigma| = \sigma$, and $L = L(G)$. Furthermore, assume $\iota_\exists(L)$ is finite. There exists a string w of L with $\iota(w) = \iota_\exists(L)$ such that the derivation tree of w has depth at most $4n\sigma$.*

Proof. Let $w_0 \in L$ be a string such that $\iota(w_0) = \iota_\exists(L)$, and let T_0 be its derivation tree. Assume that T_0 has depth greater than $4n\sigma$. Then there exists a simple-path p in T_0 from the root to a leaf of length at least $4n\sigma + 1$ (i.e., contains $4n\sigma + 2$ nodes on it). By the pigeonhole-principle, there is one non-terminal $A \in V$ which occurs at least 4σ times on this path. Therefore, there exists the derivation $S \Rightarrow^* v_0 A v_0' \Rightarrow^* v_0 v_1 A v_1' v_0' \Rightarrow^* \ldots \Rightarrow^* v_0 v_1 \cdots v_{4\sigma-1} A v_{4\sigma-1}' \cdots v_1' v_0' \Rightarrow^* v_0 v_1 \cdots v_{4\sigma-1} w_0' v_{4\sigma-1}' \cdots v_1' v_0' = w_0$, with $v_0, v_0', \ldots, v_{4\sigma-1}, v_{4\sigma-1}', w_0' \in \Sigma^*$.

As $\iota_\exists(L)$ is finite, by Lemma 20, A has no 1-universal cycle, so $\iota(v_1 \cdots v_{4\sigma-1}) = \iota(v_{4\sigma-1}' \cdots v_1') = 0$.

We now go with i from 1 to $4\sigma - 1$ and construct a set M_ℓ as follows. For this we use the rest of the arch factorization of a word $r(\cdot)$, which is the suffix not associated with any of the arches of the respective word. We maintain a set U, which is initialized with $\text{alph}(r(v_0))$; we also initialize $M_\ell = \emptyset$. Then, when considering i, if $\text{alph}(v_i) \not\subseteq U$, we let $U \leftarrow U \cup \text{alph}(v_i)$ and $M_\ell \leftarrow M_\ell \cup \{i\}$; before moving on and repeating this procedure for $i+1$, if $U = \Sigma$, we set $U \leftarrow \emptyset$. Let us note that, during this process, because $\iota(v_1 \cdots v_{4\sigma-1}) = 0$, we set $U \leftarrow \emptyset$ at most once. Also, since M_ℓ is updated only when $\text{alph}(v_i) \not\subseteq U$, it means that M_ℓ is updated at most $2\sigma - 2$ times. So $|M_\ell| \leq 2\sigma - 2$.

Similarly, to construct a set M_r, for i from $4\sigma - 1$ downto 1, we maintain a set U, initialized with $\text{alph}(r(v_0 v_1 \cdots v_{4\sigma-1} w_0'))$; we also initialize $M_r = \emptyset$. Then, when considering i, if $\text{alph}(v_i') \not\subseteq U$, we let $U \leftarrow U \cup \text{alph}(v_i')$ and $M_r \leftarrow M_r \cup \{i\}$; before moving on and repeating this procedure for $i-1$, if $U = \Sigma$, we set $U \leftarrow \emptyset$. As before, we get that M_r is updated at most $2\sigma - 2$ times, and $|M_r| \leq 2\sigma - 2$.

It is worth noting that the indices stored in M_ℓ and M_r indicate the strings v_i and v_i', respectively, which contain letters that are relevant when computing the arch factorization of w_0. The indices not contained in these sets indicate strings v_i or v_i', respectively, which are simply contained in an arch, and all the letters of these strings already appeared in that arch before the start of v_i and v_i', respectively.

As $|M_\ell| + |M_r| \leq 4\sigma - 4$, we get that there exists $i \in [1, 4\sigma]$ such that $i \notin M_\ell \cup M_r$. It is now immediate that the derivation $S \Rightarrow^* v_0 A v'_0 \Rightarrow^* v_0 v_1 A v'_1 v'_0 \Rightarrow^* \ldots \Rightarrow^* v_0 v_1 \cdots v_{i-1} A v'_{i-1} \cdots v'_1 v'_0 \Rightarrow^* v_0 \cdots v_{i-1} v_{i+1} A v'_{i+1} v'_{i-1} \cdots v'_0 \Rightarrow^* v_0 \cdots v_{i-1} v_{i+1} \cdots v_{4\sigma-1} w'_0 v'_{4\sigma-1} \cdots v'_{i+1} v'_{i-1} \cdots v'_0 = w_1$ produces a string w_1 such that $\iota(w_1) = \iota(w_0)$; let T_1 be the tree corresponding to this derivation. Clearly, the total length of the simple-paths connecting the root to leaves in the derivation tree T_1 is strictly smaller than the total length of the simple-paths connecting the root to leaves in the tree T_0. If T_1 still has root-to-leaf simple-paths of length at least $4n\sigma$, we can repeat this process and obtain a tree where the total length of the simple-paths connecting the root to leaves is even smaller. This process is repeated as long as we obtain trees having at least one root-to-leaf simple-path of length at least $4n\sigma$. Clearly, this is a finite process, whose number of iterations is bounded by, e.g., the sum of the length of root-to-leaf simple-paths of T_0. When we obtain a tree T where all root-to-leaf simple paths are of length at most $4n\sigma$, we stop and note that the border of this tree is a string w, with $\iota(w) = \iota_\exists(L)$. This concludes our proof. ◀

We now come to the algorithmic consequences of our combinatorial lemmas. For both considered problems the language given as input is in the form of a CFG $G = (V, \Sigma, P, S)$ in CNF with $|\Sigma| = \sigma \geq 2$ and $|V| = n$. Firstly, we show that Problem 5 can be decided in polynomial time.

▶ **Theorem 22.** *Problem 5 can be solved in $\mathcal{O}(\max(n^3, n^2\sigma))$ time.*

Proof Sketch. By Lemma 20, it is enough to check whether G contains a non-terminal $X \in V$ such that X has a 1-universal cycle. More precisely, we want to check if there exists a non-terminal X such that $X \Rightarrow^* wXw'$, where $\text{alph}(w) = \Sigma$ or $\text{alph}(w') = \Sigma$. We only show how to decide if there is a non-terminal X such that $X \Rightarrow^* wXw'$, where $\text{alph}(w) = \Sigma$ (the case when $\text{alph}(w') = \Sigma$ is similar). The main observation is that such a non-terminal $X \in V$ exists if and only if G contains, for some non-terminal X, derivations $X \Rightarrow^* w_a X w'_a$, with $w_a \in \Sigma^* a \Sigma^*$ and $w'_a \in \Sigma^*$, for all $a \in \Sigma$. Determining the existence of such a non-terminal is done in several steps. Firstly, we identify in $\mathcal{O}(n^3)$-time all pairs of non-terminals $A, B \in V$ with $A \Rightarrow^* \alpha B \beta$, for some $\alpha, \beta \in \Sigma^*$. Then, using the previously computed pairs, in $\mathcal{O}(n^2\sigma)$, we identify all pairs A, a, with $A \in V$ and $a \in \Sigma$, with $A \Rightarrow^* \alpha a \beta$, for some $\alpha, \beta \in \Sigma^*$. Now, in $\mathcal{O}(n^3)$ time, we identify all pairs of non-terminals $A, B \in V$, such that there exist a production $A \to BC$ in G and a derivation $C \Rightarrow^* \alpha A \beta$, with $\alpha, \beta \in \Sigma^*$. Finally, using all the sets of pairs that we have computed, we can identify all pairs of A, a, of non-terminals and terminals of G, respectively, such that there exists a derivation $A \Rightarrow^* \alpha a \beta A \gamma$, with $\alpha, \beta, \gamma \in \Sigma^*$. We conclude that there exists a non-terminal $X \in V$ for which we have derivations $X \Rightarrow^* w_a X w'_a$, with $w_a \in \Sigma^* a \Sigma^*$ and $w'_a \in \Sigma^*$, for all $a \in \Sigma$, if and only if there exists such a non-terminal X where the pairs X, a were found in the last step of our approach, for all $a \in \Sigma$. ◀

Further, we show that Problem 3 is FPT w.r.t. the parameter σ; this also means that the respective problem is solvable in polynomial time for constant-size alphabets. Recall that there is an ETH-conditional lower bound of $2^{o(\sigma)}\text{poly}(n, \sigma)$ for the time complexity of algorithms solving this problem.

▶ **Theorem 23.** *Problem 3 can be solved in $\mathcal{O}(2^{4\sigma} n^5 \sigma^2)$ time.*

Proof Sketch. Recall that now we also get as input an integer $k > 0$ (given in binary representation).

To solve Problem 3, we check, using the algorithm from Theorem 22, whether $\iota_\exists(L)$ is finite. If $\iota_\exists(L)$ is infinite, then we answer the given instance positively. Otherwise, we proceed as follows.

We use a dynamic programming approach to compute the maximal universality index of a string of L. This essentially uses the result of Lemma 21 which states that such a string is the border of a derivation tree of depth at most $N = 4n\sigma$. More precisely, we construct a 4-dimensional matrix $M[\cdot,\cdot,\cdot,\cdot]$, with elements $M[i, A, S_p^A, S_s^A]$ with $A \in V$, $S_p^A, S_s^A \subsetneq \Sigma$ and $i \leq N$. By definition, $M[i, A, S_p^A, S_s^A] = \ell$ if ℓ is the maximum number with the property that there exists a string w, which labels the border of a derivation tree of height at most i rooted in A, so that w has a prefix x, with $\mathrm{alph}(x) = S_p^A$, followed by ℓ arches, and a suffix y with $\mathrm{alph}(y) = S_s^A$.

To compute the elements $M[i,\cdot,\cdot,\cdot]$ for $i = 1$ it is enough to consider the productions of the form $A \to a$. For each such production, we only have to set $M[1, A, \{a\}, \emptyset] \leftarrow 0$ and $M[1, A, \emptyset, \{a\}] \leftarrow 0$.

To compute $M[i,\cdot,\cdot,\cdot]$ for $i > 1$, we consider every production $A \to BC$, and try to combine derivation trees of height at most $i-1$ and obtain derivation trees of height i. This computation is structured in two phases (corresponding to two cases).

The first phase corresponds to first of the cases we need to consider. Namely, in this case, we produce trees of height at most i whose borders have 0 arches, by combining trees of height at most $i-1$ with the same property. For that, we iterate over the productions $A \to BC$, and sets $S_1, S_2, S_3, S_4 \subsetneq \Sigma$ such that $M[i-1, B, S_1, S_2] = 0$ and $M[i-1, C, S_3, S_4] = 0$. If $S_1 \cup S_2 \cup S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2 \cup S_3 \cup S_4, \emptyset] = 0$ and $M[i, A, \emptyset, S_1 \cup S_2 \cup S_3 \cup S_4] = 0$. If $S_1 \cup S_2 \cup S_3 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2 \cup S_3, S_4] = 0$. If $S_1 \cup S_2 \subsetneq \Sigma$ and $S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1 \cup S_2, S_3 \cup S_4] = 0$. Finally, if $S_2 \cup S_3 \cup S_4 \subsetneq \Sigma$, we set $M[i, A, S_1, S_2 \cup S_3 \cup S_4] = 0$. Note that the elements of $M[i,\cdot,\cdot,\cdot]$ set in this step might still be updated in the following. Moreover, the case when we can join two trees of height at most $i-1$ whose borders have 0 arches, and obtain a tree of height i whose border has one arch is also considered in the following.

The second case (and corresponding phase of our computation) is, therefore, the one where we produce trees of height at most i whose borders have at least one arch, by combining trees of height at most $i-1$. In this case, we iterate over the productions $A \to BC$, and sets $S_p^A, S_s^A \subsetneq \Sigma$. Now, for every pair $R, R' \subsetneq \Sigma$, with $R \cup R' = \Sigma$, a possible candidate for $M[i, A, S_p^A, S_s^A]$, corresponding to the production $A \to BC$, is obtained by adding $M[i-1, B, S_p^A, R]$ and $M[i-1, C, R', S_s^A]$, and add one for the new arch. We then take the maximum over all these combinations of alphabets R and R'. We get

$$c_{A\to BC} \leftarrow \max(\{-\infty\} \cup \{M[i-1, B, S_p^A, R] + M[i-1, C, R', S_s^A] + 1 \mid R, R' \subsetneq \Sigma, R \cup R' = \Sigma\}).$$

If A has t productions $p_1, \ldots, p_t$ we compute all values $c_{p_1}, \ldots, c_{p_t}$. Then $M[i, A, S_p^A, S_s^A]$ is set to be the maximum of the current value of $M[i, A, S_p^A, S_s^A]$ (as potentially computed in the first phase), $c_{p_1}, \ldots, c_{p_t}$, and $M[i-1, A, S_p^A, S_s^A]$.

For each i, this process (covering both cases) requires $\mathcal{O}(n^3 2^{4\sigma})$ algorithm-steps, in the worst case. So the entire matrix M is computed in $\mathcal{O}(2^{4\sigma} n^4 \sigma)$ algorithm-steps, where each algorithm-step might require $\mathcal{O}(n\sigma)$-time (as it can involve arithmetical operations on numbers with $\mathcal{O}(n\sigma)$ bits). We obtain, in the end, the complexity from the statement. Note that the matrix $M[\cdot,\cdot,\cdot,\cdot]$ computed by our algorithm has $O(2^{2\sigma} n^2)$ entries, so the space used by our algorithm is exponential.

Then, $\iota_\exists(L)$ equals the maximum over the entries of $M[4n\sigma, S, \emptyset, R]$, over all subsets $R \subsetneq \Sigma$, as we only consider strings w that lie in L, so strings that can be derived from S. The answer of the given instance of Problem 3 is positive if and only if $k \leq \iota_\exists(L)$. ◀

The algorithm from Theorem 23 uses exponential space (due to the usage of the matrix M). However, there is also a simple (non-deterministic) PSPACE-algorithm solving this problem. Such an algorithm constructs non-deterministically the left derivation (where, at each step, the leftmost non-terminal is rewritten) of a string $w \in L$ with $\iota(w) \geq k$; w is non-deterministically guessed, and it is never constructed or stored explicitly by our algorithm. During this derivation of w the number of non-terminals in each sentential form is upper bounded by the depth of its derivation tree [35]; due to Lemma 21, we thus can have only $4n\sigma$ such non-terminals (if this number becomes larger, we stop and reject: the derivation tree of the guessed derivation is too deep for our purposes). During the simulation of the leftmost derivation, at step i, we also do not keep track of the maximal prefix w'_i consisting only of terminals of the sentential form, but only of $\iota(w'_i)$, $\mathrm{alph}(r(w'_i))$, and of the maximal suffix w''_i consisting of non-terminals only (i.e., the part we still need to process); this is enough for computing the universality of the derived string. The information stored by our algorithm clearly fits in polynomial space. If, and only if, at the end of the derivation, the maintained universality index is at least k, we accept the input grammar and number k.

6 Problem 4

Let us note that deciding Problem 4 for some input language L and integer k is equivalent to deciding whether $\iota_\forall(L) \geq k$. In [4], it was shown that for a regular language L over an alphabet with σ letters, accepted by an NFA with s states, Problem 4 can be decided in $\mathcal{O}(s^3\sigma)$.

For the rest of this section, we consider Problem 4 for the class CFL, and we assume that we are given a CFL L by a CFG G in CNF, with n non-terminals, over an alphabet Σ, with $\sigma \geq 2$ letters. Recall that our approach is to compute $\iota_\forall(L)$ and compare it with the input integer k.

As before, we start with a combinatorial observation. Intuitively, when we try to find a word with the lowest universality index, it is enough to consider words w, whose derivation trees do not contain root-to-leaf paths which contain twice the same non-terminal (otherwise, such a tree could be reduced, to a derivation tree of a word with potentially lower universality index).

▶ **Lemma 24.** *If $w \in L$ is a string with $\iota(w) \leq \iota(w')$, for all $w' \in L$, then there exists a string $w'' \in L$ with $\iota(w'') = \iota(w)$ and the derivation tree of w'' has depth at most n.*

We now show that we can compute $\iota_\forall(L)$ in polynomial time, when the input language is a CFL.

▶ **Theorem 25.** *Problem 4 can be solved in $\mathcal{O}(n^4\sigma^2)$ time.*

Proof Sketch. In order to compute $\iota_\forall(L)$, it is enough to compute the smallest $\ell \in \mathbb{N}$ for which there exists $w \in L$ having an absent subsequence of length ℓ (and then we conclude that $\iota_\forall(L) = \ell - 1$).

Our approach to computing $\iota_\forall(L)$ is, therefore, to define a 4-dimensional matrix M whose elements are $M[i, A, a, b]$, with $i \in [n]$, $A \in V$, $a \in \Sigma \cup \{\varepsilon\}, b \in \Sigma$. We define $M[i, A, a, b] = \ell$ if and only if there exists a word $w \in \Sigma^*$ such that $A \Rightarrow^* w$ and this derivation has an associated tree of depth at most i, and any $\mathrm{SAS}_{a,b}(w)$ has length ℓ. Based on Lemma 24, the elements of M can be computed by dynamic programming, by considering i from 1 to n, in $\mathcal{O}(n^4\sigma^2)$.

Once all elements of M are computed, we note that $\iota_\forall(L)$ is obtained by subtracting 1 from the minimum element of the form $M[n, S, a, b]$, with $a \in \Sigma \cup \{\varepsilon\}$ and $b \in \Sigma$. ◀

7 What Next? Conclusions and First Steps Towards Future Work

A conclusion of this work is that the complexity of the approached problems is, to a certain extent, similar when the input language is from the classes REG and CFL and they all become undecidable for CSL. So, a natural question is whether there are classes of languages (defined by corresponding classes of grammars or automata) between REG and CSL which exhibit a different, interesting behaviour.

We commence here this investigation by considering the class of languages accepted by a model of automata, namely, the deterministic finite automata with translucent letters (or, for short, translucent (finite) automaton – TFA), which generalizes the classical DFA by allowing the processing of the input string in an order which is not necessarily the usual sequential left-to-right order (without the help of an explicit additional storage unit). These automata, first considered in [60] (see also the survey [63] for a discussion on their properties and motivations), are strictly more powerful than classical finite automata and are part of a class of automata-models that are allowed to jump symbols in their processing, e.g., see [58] or [15]. From our perspective, these automata and the class of languages they accept are interesting because, on the one hand, they seem to be a generalization of regular languages which is orthogonal to the classical generalization provided by context-free languages, and, on the other hand, initial results suggest that the problems considered in this paper, not only become harder for them, but also their decidability fills the gap between the polynomial time solubility in the case of CFLs and that of undecidability for the class of CSL.

So, in what follows, we discuss some problems from Section 1 in relation to the TFA model, following the formalization from [59].

▶ **Definition 26.** *A TFA M is a tuple $M = (Q, \Sigma, q_0, F, \delta)$, just as in the case of DFA. However, the processing of inputs is not necessarily sequential. We define the partial relation $\circlearrowright$ on the set $Q \times \Sigma^*$ of configurations of M: $(p, xay) \circlearrowright_M (q, xy)$ if $\delta(p, a) = q$, and $\delta(p, b)$ is not defined for any $b \in alph(x)$, where $p, q \in Q$, $a, b \in \Sigma$, $x, y \in \Sigma^*$. The subscript M is omitted when it is understood from the context. The reflexive and transitive closure of $\circlearrowright$ is $\circlearrowright^*$ and the language accepted by M is defined as $L(M) = \{w \in \Sigma^* \mid (q_0, w) \circlearrowright^* (f, \varepsilon) \text{ for some } f \in F\}$.*

In this model, letters a such that $\delta(p, a)$ is not defined are called translucent for p, hence the name of the model. The machine reads and erases from the tape the letters of the input one-by-one. Note that the definition requires that every letter of the input is read before it can be accepted. This is slightly different from the original definition [60], which did not require all of the letters read, and used an unerasable endmarker on the tape. TFA by our definition can be trivially simulated by a machine with the original definition, and our results stand for the original model, too. We chose to follow the definitions in [59], because in our opinion it is simpler (and simpler to argue), and illustrates the difficulty of the subsequence matching problems for nonsequential machine models just as well.

A first observation is that, in terms of execution, in each step a TFA reads (and consumes) the leftmost unconsumed symbol which allows a transition (i.e., that has not been previously read, and there is a transition labeled with it from the current state). Therefore, for every individual letter, the order of the processing of its occurrences in the TFA is that in which they appear in a string. The non-deterministic version of this automata model accepts all rational trace languages, and all accepted languages have semi-linear Parikh images. Moreover, the class of languages accepted by this model is incomparable to the class of CFL, while still being CS. The class of languages accepted by the more restrictive deterministic finite automata with translucent letters, for short TFA, strictly includes the class REG and

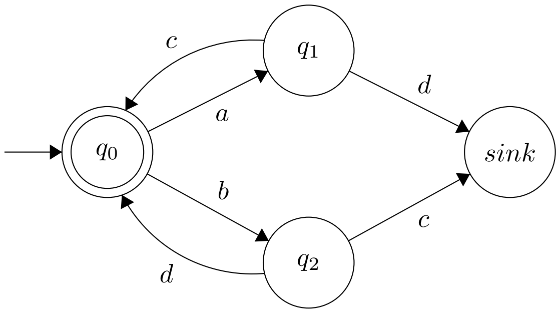


Figure 1 TFA that accepts the language $w \sqcup\!\sqcup h(w)$, where $w \in \{a, b\}^*$ and h is a morphism of the form $h(a) = c$, $h(b) = d$.

is still incomparable with CFL and the above mentioned class of rational trace languages. The recent survey [63] overviews the extensive literature regarding variations of these types of machines.

▶ **Example 27.** The TFA in Figure 1 accepts the language $L = w \sqcup\!\sqcup h(w)$, where $w \in \{a, b\}^*$ and $h : \{a, b\}^* \to \{c, d\}^*$ is a morphism given by $h(a) = c$, $h(b) = d$. Here $\sqcup\!\sqcup$ denotes the usual shuffle operation for words over some alphabet Σ, i.e., $u \sqcup\!\sqcup v = \{u_1v_1 \cdots u_\ell v_\ell \mid u = u_1 \cdots u_\ell, v = v_1 \cdots v_\ell, u_i \in \Sigma^*$ for $i \in [\ell], v_i \in \Sigma^*$ for $i \in [\ell]\}$; in our case, $\Sigma = \{a, b, c, d\}$.

Coming back to the TFA in Figure 1: in state q_0, the machine can read only the first a or b remaining on the tape and immediately matches it with the first c or d, respectively. If it reads a and in the remaining input the first d comes before the first c, it goes into the sink state. Similarly, if it reads b but the first remaining c is before the first remaining d, it goes to sink, because the projection of the input to the $\{a, b\}$ alphabet does not match the projection to the $\{c, d\}$ alphabet. The language L is not context-free. This can be, indeed, seen by intersecting it with the regular language $(a + b)^*(c + d)^*$, which yields the language $\{w \cdot h(w) \mid w \in \{a, b\}^*\}$, a variant of the so called "copy language". This language is non-context-free, by an easy application of the Bar-Hillel pumping lemma, so L is not context-free.

We first note that the class of languages accepted by TFA becomes incomparable to that of CFLs only starting from the ternary alphabet case (see [61]), since, for a TFA over a binary alphabet, one can construct a push-down automaton accepting the same language.

▶ **Theorem 28.** *The languages accepted by TFA over binary alphabets are CF.*

As a consequence of this and of the results shown in the previous sections we get the following.

▶ **Theorem 29.** *Over binary alphabets, all problems of Section 1 are decidable, and except for Problem 3, all are decidable in polynomial time for a TFA A given as input.*

Thus, our interest now shifts to languages accepted by TFAs, over alphabets Σ of size $\sigma \geq 3$. We report here a series of initial results, which suggest this to be a worthwhile direction of investigation. We first note that we cannot apply the approach from the general Theorem 15 to solve the problems considered, since one can encode the solution set of any Post Correspondence Problem (for short, PCP) instance as the intersection of a regular language and a language accepted by a TFA. This is a first significant difference w.r.t. the status of the approached problems for the case of REG and CFL.

▶ **Theorem 30.** *The emptiness problem for languages defined as the intersection of the language accepted by a TFA with a regular language (given as finite automaton) is undecidable.*

The decidability of Problem 1 for larger alphabets in the case of TFA is settled by an exponential time brute force algorithm, after establishing that if the input language contains a supersequence of w, then it also contains one whose length is bounded by a polynomial in the size of the input. By a reduction from the well-known NP-complete Hamiltonian Cycle Problem [27], we can also show that Problem 1 for TFA is NP-hard over unbounded alphabets (containment in NP follows from the same length upper bound mentioned earlier). This is again a significant deviation w.r.t. the status of this problem for the case when the input language is given by a finite automaton or by a CFG.

▶ **Theorem 31.** *Problem 1 is NP-complete over unbounded alphabets.*

Since our initial results deviate from the corresponding results obtained for CFL, without suggesting that the considered problems become undecidable, completing this investigation for all other problems seems worthwhile to us. While we have excluded the approach from the general Theorem 15, we cannot yet say anything about the approach in Theorem 14. It remains an interesting open problem (also of independent interest w.r.t. to our research) to obtain an algorithm for computing the downward closure of a TFA-language, or show that such an algorithm does not exist.

While studying the problems discussed in this paper for TFAs seems an interesting way to understand their possible further intricacies, which cause the huge gap between their status for CFL and CSL, respectively, another worthwhile research direction is to consider them in the context of other well-motivated classes of languages, for which all these problems are decidable, and try to obtain optimised algorithms in those cases.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018. doi:10.4230/LIPIcs.ITCS.2018.35.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014. doi:10.1007/978-3-662-43948-7_4.

4 Duncan Adamson, Pamela Fleischmann, Annika Huch, Tore Koß, Florin Manea, and Dirk Nowotka. k-universality of regular languages. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *LIPIcs*, pages 4:1–4:21, 2023. Full version: https://arxiv.org/abs/2311.10658. doi:10.4230/LIPIcs.ISAAC.2023.4.

5 Duncan Adamson, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Longest common subsequence with gap constraints. In Anna E. Frid and Robert Mercas, editors, *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13899 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 2023. doi:10.1007/978-3-031-33180-0_5.

6 Ashwani Anand and Georg Zetzsche. Priority downward closures. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.39.

7 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 7–10, 2017. doi:10.1145/3093742.3095106.

8 Ricardo A. Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78(2):363–376, 1991. doi:10.1016/0304-3975(91)90358-9.

9 Laura Barker, Pamela Fleischmann, Katharina Harwardt, Florin Manea, and Dirk Nowotka. Scattered factor-universality of words. In Nataša Jonoska and Dmytro Savchuk, editors, *Developments in Language Theory*, LNCS, pages 14–28, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-48516-0_2.

10 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In Pablo de la Fuente, editor, *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000*, pages 39–48, , 2000. doi:10.1109/SPIRE.2000.878178.

11 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theoretical Computer Science*, 443:25–34, 2012. doi:10.1016/j.tcs.2012.03.029.

12 Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, Streaming, and Fine-Grained Complexity of (Weighted) LCS. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *LIPIcs*, pages 40:1–40:16, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.40.

13 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. SODA 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.

14 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *Journal of Computer and System Sciences*, 80(4):766–776, 2014. doi:10.1016/j.jcss.2013.11.002.

15 Hiroyuki Chigahara, Szilárd Zsolt Fazekas, and Akihiro Yamamura. One-way jumping finite automata. *International Journal of Foundations of Computer Science*, 27(3):391–405, 2016.

16 Vaclav Chvatal and David Sankoff. Longest common subsequences of two random sequences. *Journal of Applied Probability*, 12(2):306–315, 1975. URL: http://www.jstor.org/stable/3212444.

17 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

18 Maxime Crochemore, Borivoj Melichar, and Zdeněk Troníček. Directed acyclic subsequence graph — overview. *Journal of Discrete Algorithms*, 1(3-4):255–280, 2003. doi:10.1016/S1570-8667(03)00029-7.

19 Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. The Edit Distance to k-Subsequence Universality. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *LIPIcs*, pages 25:1–25:19, 2021. doi:10.4230/LIPIcs.STACS.2021.25.

20 Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 64:1–64:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.64.

21 Lukas Fleischer and Manfred Kufleitner. Testing Simon's congruence. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPIcs*, pages 62:1–62:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.62.

22 Pamela Fleischmann, Sungmin Kim, Tore Koß, Florin Manea, Dirk Nowotka, Stefan Siemer, and Max Wiedenhöft. Matching patterns with variables under simon's congruence. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems - 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2023. doi:10.1007/978-3-031-45286-4_12.

23 Michael L. Fredman and Dan E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 1–7. ACM, 1990. doi:10.1145/100216.100217.

24 Dominik D. Freydenberger, Pawel Gawrychowski, Juhani Karhumäki, Florin Manea, and Wojciech Rytter. Testing k-binomial equivalence. *https://arxiv.org/abs/1509.00622*, pages 239–248, 2015.

25 André Frochaux and Sarah Kleest-Meißner. Puzzling over subsequence-query extensions: Disjunction and generalised gaps. In Benny Kimelfeld, Maria Vanina Martinez, and Renzo Angles, editors, *Proceedings of the 15th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2023), Santiago de Chile, Chile, May 22-26, 2023*, volume 3409 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3409/paper3.pdf.

26 Emmanuelle Garel. Minimal separators of two words. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, 4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993, Proceedings*, volume 684 of *LNCS*, pages 35–53. Springer, 1993. doi:10.1007/BFB0029795.

27 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA, 1990.

28 Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 34:1–34:18, 2021. doi:10.4230/LIPIcs.STACS.2021.34.

29 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *The VLDB Journal*, 29(1):313–352, 2020. doi:10.1007/s00778-019-00557-w.

30 Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the size of Higman-Haines sets: Effective constructions. *Fundamenta Informaticae*, 91(1):105–121, 2009. doi:10.3233/FI-2009-0035.

31 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17, pages 1–12, 2017. doi:10.1109/LICS.2017.8005141.

32 Jean-Jacques Hébrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82:35–49, 1991.

33 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

34 Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977. doi:10.1145/322033.322044.

35 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

36 James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Communications of the ACM*, 20(5):350–353, 1977. doi:10.1145/359581.359603.

37 P. Karandikar, M. Kufleitner, and P. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015. doi:10.1016/J.IPL.2014.11.008.

38 Prateek Karandikar and Philippe Schnoebelen. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:6)2019.

39 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In Dan Olteanu and Nils Vortmeier, editors, *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, volume 220 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICDT.2022.18.

40 Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. Discovering multi-dimensional subsequence queries from traces - from theory to practice. In Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen, editors, *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*, volume P-331 of *LNI*, pages 511–533. Gesellschaft für Informatik e.V., 2023. doi:10.18420/BTW2023-24.

41 Maria Kosche, Tore Koß, Florin Manea, and Viktoriya Pak. Subsequences in bounded ranges: Matching and analysis problems. In Anthony W. Lin, Georg Zetzsche, and Igor Potapov, editors, *Reachability Problems - 16th International Conference, RP 2022, Kaiserslautern, Germany, October 17-21, 2022, Proceedings*, volume 13608 of *Lecture Notes in Computer Science*, pages 140–159. Springer, 2022. doi:10.1007/978-3-031-19135-0_10.

42 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Absent subsequences in words. In Paul C. Bell, Patrick Totzke, and Igor Potapov, editors, *Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings*, volume 13035 of *LNCS*, pages 115–131, 2021. doi:10.1007/978-3-030-89716-1_8.

43 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Absent subsequences in words. *Fundam. Informaticae*, 189(3-4):199–240, 2022. doi:10.3233/FI-222159.

44 Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Combinatorial algorithms for subsequence matching: A survey. In Henning Bordihn, Géza Horváth, and György Vaszil, editors, *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, volume 367 of *EPTCS*, pages 11–27, , 2022. doi:10.4204/EPTCS.367.2.

45 Dietrich Kuske. The subtrace order and counting first-order logic. In Henning Fernau, editor, *Computer Science – Theory and Applications*, volume 12159 of *LNCS*, pages 289–302, 2020. doi:10.1007/978-3-030-50026-9_21.

46 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures*, LNCS, pages 348–364, 2019. doi:10.1007/978-3-030-17127-8_20.

47 Martin Lange and Hans Leiß. To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm. *Informatica Didact.*, 8, 2009. URL: http://ddi.cs.uni-potsdam.de/InformaticaDidactica/LangeLeiss2009.

48 Marie Lejeune, Julien Leroy, and Michel Rigo. Computing the k-binomial complexity of the Thue-Morse word. In Piotrek Hofman and Michał Skrzypczak, editors, *Developments in Language Theory*, LNCS, pages 278–291, 2019. doi:10.1007/978-3-030-24886-4_21.

49 Julien Leroy, Michel Rigo, and Manon Stipulanti. Generalized Pascal triangle for binomial coefficients of words. *The Electronic Journal of Combinatorics*, 24(1.44):36 pp., 2017.

50 Chun Li and Jianyong Wang. *Efficiently Mining Closed Subsequences with Gap Constraints*, pages 313–322. SIAM, 2008. doi:10.1137/1.9781611972788.28.

51 Chun Li, Qingyan Yang, Jianyong Wang, and Ming Li. Efficient mining of gap-constrained subsequences and its various applications. *ACMTransactions on Knowledge Discovery from Data*, 6(1):2:1–2:39, 2012. doi:10.1145/2133360.2133362.

52 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/GCC-2012-0016.

53 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

54 David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, April 1978. doi:10.1145/322063.322075.

55 Florin Manea, Jonas Richardsen, and Markus L. Schmid. Subsequences with generalised gap constraints: Upper and lower complexity bounds. In Shunsuke Inenaga and Simon J. Puglisi, editors, *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CPM.2024.22.

56 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

57 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Subword histories and Parikh matrices. *Journal of Computer and System Sciences*, 68(1):1–21, 2004. doi:10.1016/J.JCSS.2003.04.001.

58 Alexander Meduna and Petr Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012. doi:10.1142/S0129054112500244.

59 Victor Mitrana, Andrei Păun, Mihaela Păun, and José-Ramón Sánchez-Couso. Jump complexity of finite automata with translucent letters. *Theoretical Computer Science*, 992:114450, 2024. doi:10.1016/J.TCS.2024.114450.

60 Benedek Nagy and Friedrich Otto. Finite-state acceptors with translucent letters. In A.O. De La Puente G. Bel-Enguix, V. Dahl, editor, *BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology*, pages 3–13, 2011.

61 Benedek Nagy and Friedrich Otto. Globally deterministic CD-systems of stateless R-automata with window size 1. *International Journal of Computer Mathematics*, 90(6):1254–1277, 2013. doi:10.1080/00207160.2012.688820.

62 Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982. doi:10.1007/BF00264437.

63 Friedrich Otto. A survey on automata with translucent letters. In Benedek Nagy, editor, *Implementation and Application of Automata*, pages 21–50, Cham, 2023. doi:10.1007/978-3-031-40247-0_2.

64 Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

65 William E. Riddle. An approach to software system modelling and analysis. *Computer Languages*, 4(1):49–66, 1979. doi:10.1016/0096-0551(79)90009-2.

66 Michel Rigo and Pavel Salimov. Another generalization of abelian equivalence: Binomial complexity of infinite words. *Theoretical Computer Science*, 601:47–57, 2015. doi:10.1016/J.TCS.2015.07.025.

67 Arto Salomaa. Connections between subwords and certain matrix mappings. *Theoretical Computer Science*, 340(2):188–203, 2005. doi:10.1016/J.TCS.2005.03.024.

68 Philippe Schnoebelen and Julien Veron. On arch factorization and subword universality for words and compressed words. In Anna Frid and Robert Mercaş, editors, *Combinatorics on Words*, volume 13899, pages 274–287, 2023. doi:10.1007/978-3-031-33180-0_21.

69 Shinnosuke Seki. Absoluteness of subword inequality is undecidable. *Theoretical Computer Science*, 418:116–120, 2012. doi:10.1016/J.TCS.2011.10.017.

70 Alan C. Shaw. Software descriptions with flow expressions. *IEEE Transactions on Software Engineering*, 4(3):242–254, 1978. doi:10.1109/TSE.1978.231501.

71 Imre Simon. *Hierarchies of events with dot-depth one - Ph.D. thesis*. University of Waterloo, 1972.

72 Imre Simon. Piecewise testable events. In H. Brakhage, editor, *Automata Theory and Formal Languages*, LNCS, pages 214–222, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. doi:10.1007/3-540-07407-4_23.

73 Imre Simon. Words distinguished by their subwords (extended abstract). In *Proc. WORDS 2003*, volume 27 of *TUCS General Publication*, pages 6–13, 2003.

74 Zdeněk Troníček. Common subsequence automaton. In Jean-Marc Champarnaud and Denis Maurel, editors, *Conference on Implementation and Application of Automata*, volume 2608 of *LNCS*, pages 270–275, 2002. doi:10.1007/3-540-44977-9_28.

75 Jan van Leeuwen. Effective constructions in well-partially- ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978. doi:10.1016/0012-365X(78)90156-5.

76 Georg Zetzsche. The Complexity of Downward Closure Comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 123:1–123:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.

77 Georg Zetzsche. Separability by piecewise testable languages and downward closures beyond subwords. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 929–938. ACM, 2018. doi:10.1145/3209108.3209201.

78 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228, 2014. doi:10.1145/2588555.2593671.

Coordinated Motion Planning: Multi-Agent Path Finding in a Densely Packed, Bounded Domain

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany

Ramin Kosfeld
Department of Computer Science, TU Braunschweig, Germany

Peter Kramer
Department of Computer Science, TU Braunschweig, Germany

Jonas Neutzner
Department of Computer Science, TU Braunschweig, Germany

Christian Rieck
Department of Computer Science, TU Braunschweig, Germany

Christian Scheffer
Department of Electrical Engineering and Computer Science,
Bochum University of Applied Sciences, Germany

Abstract

We study Multi-Agent Path Finding for arrangements of labeled agents in the interior of a simply connected domain: Given a unique start and target position for each agent, the goal is to find a sequence of parallel, collision-free agent motions that minimizes the overall time (the *makespan*) until all agents have reached their respective targets. A natural case is that of a simply connected polygonal domain with axis-parallel boundaries and integer coordinates, i.e., a *simple polyomino*, which amounts to a simply connected union of lattice unit squares or *cells*. We focus on the particularly challenging setting of densely packed agents, i.e., one per cell, which strongly restricts the mobility of agents, and requires intricate coordination of motion.

We provide a variety of novel results for this problem, including (1) a characterization of polyominoes in which a reconfiguration plan is guaranteed to exist; (2) a characterization of shape parameters that induce worst-case bounds on the makespan; (3) a suite of algorithms to achieve asymptotically worst-case optimal performance with respect to the achievable *stretch* for cases with severely limited maneuverability. This corresponds to bounding the ratio between obtained makespan and the lower bound provided by the max-min distance between the start and target position of any agent and our shape parameters.

Our results extend findings by Demaine et al. [13, 14] who investigated the problem for solid rectangular domains, and in the closely related field of Permutation Routing, as presented by Alpert et al. [6] for convex pieces of grid graphs.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases multi-agent path finding, coordinated motion planning, bounded stretch, makespan, swarm robotics, reconfigurability, parallel sorting

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.29

Related Version *Full Version*: `https://arxiv.org/abs/2409.06486` [18]

Funding This work was partially supported by the German Research Foundation (DFG), project “Space Ants”, FE 407/22-1 and SCHE 1931/4-1.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 29; pp. 29:1–29:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Problems of coordinating the motion of a set of objects occur in a wide range of applications, such as warehouses [32], multi-agent motion planning [27, 28], and aerial swarm robotics [11]. In Multi-Agent Path Finding (MAPF) [31], we are given a set of agents, each with an initial and a desired target position within a certain domain. The task is to determine a coordinated motion plan: a sequence of parallel, collision-free movements such that the time by which all agents have reached their destinations (the *makespan*) is minimized.

Theoretical aspects of MAPF have enjoyed significant attention. In the early days of computational geometry, Schwartz and Sharir [29] developed methods for coordinating the motion of disk-shaped objects between obstacles, with runtime polynomial in the complexity of the obstacles, but exponential in the number of disks. The fundamental difficulty of geometric MAPF was highlighted by Hopcroft et al. [21, 22], who showed that it is PSPACE-complete to decide whether multiple agents can reach a given target configuration. In contrast, closely related graph-based variants of the MAPF problem permit for linear time algorithms for the same decision problem [35].

More recently, Demaine et al. [10, 13, 14] have provided methods to compute *constant stretch* solutions for coordinated motion planning in unbounded environments in which agents occupy distinct grid cells. The stretch of a solution is defined as the ratio between its makespan and a trivial lower bound, the *diameter* d, which refers to maximum distance between any agent's origin and destination. Their work therefore obtains collision-free motion schedules that move each agent to its target position in $\mathcal{O}(d)$ discrete moves, which corresponds to a constant-factor approximation. However, their methods assume the absence of a environmental boundary that may impose external constraints on the agents' movements.

1.1 Our contributions

In this paper, we study Multi-Agent Path Finding for densely packed arrangements of labeled agents that are required to remain within discrete grid domains, i.e., polyominoes. This is a natural constraint that occurs in many important applications, but provides considerable additional difficulties; in particular, a coordinated motion plan may no longer exist for domains with narrow bottlenecks. We provide a variety of novel contributions:

- We give a full characterization of simple polyominoes P that are universally reconfigurable. These allow *some* feasible coordinated motion plan for *any* combination of initial and desired target configurations, without regard for the makespan: We prove that this is the case if and only if P has a cover by 2×2 squares with a connected intersection graph.
- We model the shape parameters *bottleneck length* $\zeta(P)$ (which is the minimum length of a cut dividing the region into non-trivial pieces) and *domain depth* $\mu(P)$ (which is the maximum distance of any cell from the domain boundary). We provide refined upper and lower bounds on the makespan and stretch factor based on these shape parameters.
- For some instances, any applicable schedule may require a makespan of $\Omega(d + {}^{d^2}/_{\zeta(P)})$. We show how to compute schedules of makespan linear in the ratio of domain area and bottleneck, i.e., $\mathcal{O}({}^{n}/_{\zeta(P)})$.
- We characterize *narrow* instances, which feature very limited depth relative to the diameter d, and provide an approach for asymptotically worst-case optimal schedules.

Note that (parts of) the proofs of statements marked with ($\star$) have been omitted in the main sections due to space constraints; we refer the reader to the full version [18] instead.

1.2 Related work

Motion planning. Multi-Agent Path Finding is a widely studied problem. Due to space constraints, we restrict our description to the most closely related work. For more detailed references, refer to the extensive bibliography in [14] and the mentioned surveys [11, 28, 31].

Of fundamental importance to our work are the results by Demaine et al. [14], who achieved reconfiguration with constant stretch for the special case of rectangular domains. A key idea is to consider a partition of the rectangle into tiles whose size is linear in diameter d. These tiles can then be reconfigured in parallel. First, flow techniques are applied to shift agents into their target tile; afterward, agents are moved to their respective target positions. Furthermore, they showed that computing the optimal solution is strongly NP-complete.

Fekete et al. [16, 19] considered the unconstrained problem on the infinite grid with the additional condition that the whole arrangement needs to be connected after every parallel motion. They considered both the labeled and the unlabeled version of the problem, providing polynomial-time algorithms for computing schedules with constant stretch for configurations of sufficient scale. They also showed that deciding whether there is a reconfiguration schedule with a makespan of 2 is already NP-complete, unlike deciding the same for a makespan of 1.

Eiben, Ganian, and Kanj [15] investigated the parameterized complexity of the problem for the variants of minimizing the makespan and minimizing the total travel distance. They analyzed the problems with respect to two parameters: the number of agents, and the objective target. Both variants are FPT when parameterized by the number of agents, while minimizing the makespan becomes para-NP-hard when parameterized by the objective target.

Further related work studies (unlabeled) multi-robot motion planning problems in polygons. Solovey and Halperin [30] show that the unlabeled variant is PSPACE-hard, even for the specific case of unit-square robots moving amidst polygonal obstacles. Even in simple polygonal domains, a feasible motion-plan for unlabeled unit-disk robots does not always exist, if, e.g., the robots and their targets are positioned too densely. However, if there is some minimal distance separating start and target positions, Adler et al. [1] show that the problem always has a solution that can be computed efficiently. Banyassady et al. [8] prove tight separation bounds for this case. Agarwal et al. [2] consider the labeled variant with revolving areas, i.e., empty areas around start and target positions. They prove that the problem is APX-hard, even when restricting to weakly-monotone motion plans, i.e., motion plans in which all robots stay within their revolving areas while an active robot moves to its target. However, they also provide a constant-factor approximation algorithm.

The computational complexity of moving two distinguishable square-shaped robots in a polygonal environment to minimize the sum of traveled distances is still open; Agarwal et al. [3] gave the first polynomial-time $(1+\varepsilon)$-approximation algorithm.

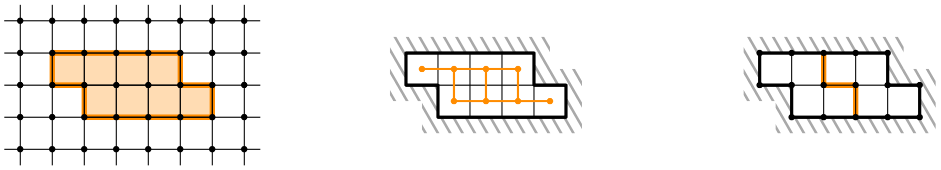
The problem was the subject of the 2021 CG:SHOP Challenge; see [17, 12, 24, 34] for an overview and a variety of practical computational methods and results.

Token swapping and routing via matchings. The task in the Token Swapping Problem is to transform two vertex labelings of a graph into one another by exchanging *tokens* between adjacent vertices by sequentially selecting individual edges. This problem is NP-complete even for trees [4], and APX-hard [26] in general. Several approximation algorithms exist for different variants and classes of graphs [20, 26, 33]. The Permutation Routing variant allows for parallelization, by selecting disjoint edge sets to perform swaps in parallel [5, 9, 23]. The *routing number* of a graph describes the maximal number of necessary parallel swaps between any two labelings. Recently, Alpert et al. [6] presented an upper bound on the routing number of convex pieces of grid graphs, which is very closely related to our setting.

1.3 Preliminaries

We define the considered motion of *agents* in a restricted environment (*domain*) as follows.

Domain. Consider the infinite integer grid graph, in which each 4-cycle bounds a face of unit area, a *cell*. Every planar edge cycle in this grid graph bounds a finite set of cells, which induces a domain that we call a *(simple) polyomino*, see Figure 1a. We exclusively consider simple polyominoes, i.e., those without holes. For the sake of readability, we might not state this for each individual polyomino in later sections. The *area* of a polyomino P is equal to the number of contained cells n. The bounding edge cycle and its incident cells are therefore called is the *boundary* and *boundary cells* of P, respectively. The dual graph of P, denoted by $G(P) = (V, E)$, has a vertex for every cell, two of which are *adjacent* if they share an edge in P, as shown in Figure 1b. The *geodesic distance* between two cells of a polyomino P corresponds to the length of a shortest path between the corresponding vertices in $G(P)$. A *cut* through a polyomino P is defined by a planar path between two vertices on the boundary of P, as shown in Figure 1c. Its cut set corresponds exactly to those edges of $G(P)$ which cross the path. Geometrically, this induces two simple subpolyominoes Q, R and write $Q, R \subset P$. We say that a cut is *trivial* if its endpoints on the boundary of P have a connecting path on the boundary that is not longer than the cut itself.

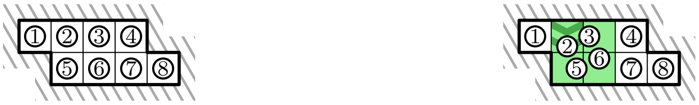


(a) A boundary cycle. **(b)** A polyomino's dual graph. **(c)** A cut through a polyomino.

Figure 1 A polyomino, its dual graph, and a cut. Subsequent illustrations will only show the boundary and any relevant cuts, foregoing the underlying integer grid.

Agents. We consider distinguishable *agents* that occupy the cells of polyominoes. A *configuration* of a polyomino P with $G(P) = (V, E)$ is a bijective mapping $C : V \to \{1, \dots, n\}$ between cells and agent labels. We denote the set of all configurations of P as $\mathcal{C}(P)$.

In each discrete time step, an agent can either *move*, changing its position v to an adjacent position w, or hold its current position. We denote this by $v \to w$ or $v \to v$, respectively. Two parallel moves $v_1 \to w_1$ and $v_2 \to w_2$ are *collision-free* if $v_1 \neq v_2$ and $w_1 \neq w_2$. We assume that a *swap*, i.e., two moves $v_1 \to v_2$ and $v_2 \to v_1$, causes a collision, and is therefore forbidden. Configurations can be *transformed* by sets of collision-free moves that are performed in parallel. If a set of moves transforms a configuration C_1 into a configuration C_2, this set is also called a *transformation* $C_1 \to C_2$. For an illustrated example, see Figure 2. A *schedule* with *makespan* $M \in \mathbb{N}$ is then a sequence of transformations $C_1 \to \cdots \to C_{M+1}$, also denoted by $C_1 \rightrightarrows C_{M+1}$.



(a) A polyomino and a configuration. **(b)** Four agents move along a cycle.

Figure 2 An illustration of an configurations and a transformations.

Problem statement. We consider the Multi-Agent Path Finding Problem for agents in a discrete environment bounded by a simple polyomino. Thus, an *instance* of the problem is composed of two configurations $C_1, C_2 \in \mathcal{C}(P)$ of a simple polyomino P. We say that a schedule is *applicable* to the instance exactly if it transforms C_1 into C_2. The *diameter of an instance* is the maximum geodesic distance d between an agents' start and target positions, and the *stretch* of an applicable schedule is the ratio between its makespan and the diameter.

2 Reconfigurability

In this section, we provide a characterization of (simple) polyominoes for which any configuration can be transformed into any other. We say that a polyomino P is *universally reconfigurable* if there exists an applicable schedule for any two configurations $C_1, C_2 \in \mathcal{C}(P)$. We prove that this is the case if and only if P has a cover by cycles that have a connected intersection graph, and show how to compute an applicable schedule of makespan $\mathcal{O}(n)$.

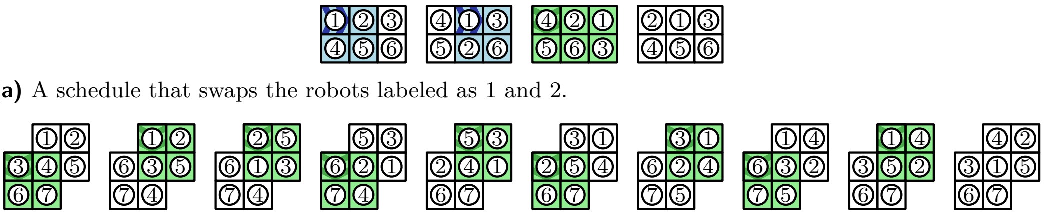
▶ **Theorem 1.** *A polyomino P is universally reconfigurable if and only if it has a cover by 2×2 squares with a connected intersection graph. For any $C_1, C_2 \in \mathcal{C}(P)$ of such a polyomino with area n, an applicable schedule $C_1 \rightrightarrows C_2$ of makespan $\mathcal{O}(n)$ can be computed efficiently.*

Due to the cyclic nature of all movement, the edge connectivity of a polyomino's dual graph plays a significant role for universal reconfigurability. We start with a negative result.

▶ **Lemma 2** (⋆). *A polyomino P that does not have a cover by 2×2 squares with a connected intersection graph is not universally reconfigurable.*

Proof sketch. We observe that transformations are inherently cyclic, as the domain is fully occupied and collisions (and thus, swaps) are forbidden. Using this fact, it is easy to show that 2-edge-connectedness is necessary and sufficient for universal reconfigurability, provided an area of at least 6 cells. From here, it is possible to show that a cover 2×2 square must exist for any two adjacent cells, implying the property. ◀

Because direct swaps of adjacent agents are not possible, an important tool is the ability to "simulate" a large number of adjacent swaps in parallel, using a constant number of transformation steps. Polyominoes that are unions of two 2×2 squares form an important primitive to achieve this. There exist two classes of such polyominoes; the squares can overlap either in one or two cells. Clearly, either dual graph can be covered by two 4-cycles that intersect in at least one vertex. In Figure 3, we illustrate schedules that perform adjacent swaps at the intersection of these cycles, implying universal reconfigurability of both classes. In fact, any instance of either class takes at most 7 or 14 transformations, respectively.



(a) A schedule that swaps the robots labeled as 1 and 2.

(b) A schedule that swaps the robots labeled as 1 and 4.

Figure 3 In polyominoes composed of two 2×2 squares, we can realize swaps in $\mathcal{O}(1)$ steps.

▶ **Observation 3.** *Polyominoes of two overlapping 2×2 squares are universally reconfigurable.*

Using this observation for a primitive local operation, we show the following.

▶ **Lemma 4.** *For any matching in the dual graph of a universally reconfigurable polyomino, we can compute a schedule of makespan $\mathcal{O}(1)$ which swaps the agents of all matched positions.*

Proof. Let I refer to the connected intersection graph of a cover of a universally reconfigurable polyomino P by 2×2 squares, which can be computed in $\mathcal{O}(n)$. Due to Lemma 2, the vertices of each edge in the dual graph of P share a common 2×2 square in the cover.

We thus divide the edges $E(I)$ of I into 36 classes based on each edge's orientation and xy-minimal coordinates mod 3. These can be represented by $\{\uparrow, \nearrow, \rightarrow, \searrow\} \times [0,2] \times [0,2]$.

For any intersection $\{u,v\} \in E(I)$, let $R(\{u,v\})$ now be the union of the vertices covered by the squares u and v, which always corresponds exactly to one of the polyominoes outlined in Observation 3. Such a region has a bounding box no larger than 3×3, which means that the regions $R(e)$ and $R(f)$ are disjoint for any two edges e and f in a common class, allowing us to apply RotateSort in parallel to all the regions within one class.

As there are constantly many classes, we can realize the adjacent swaps induced by a matching of adjacent cells in $\mathcal{O}(1)$ transformations. ◀

Marberg and Gafni [25] propose an algorithm called RotateSort that sorts a two-dimensional $n \times m$ array within $\mathcal{O}(n+m)$ parallel steps. Demaine et al. [14] demonstrate that this algorithm can be applied geometrically, using the local swap mechanism illustrated in Figure 3a. A geometric application of RotateSort is a sequence of sets of pairwise disjoint adjacent swap operations, i.e., sets consisting of pairs of adjacent cells, where swaps can be simulated by circular rotations. As our setting is not merely restricted to rectangular domains, we extend their approach using Lemma 4. We give a constructive proof of Theorem 1 in the shape of an algorithm, as follows.

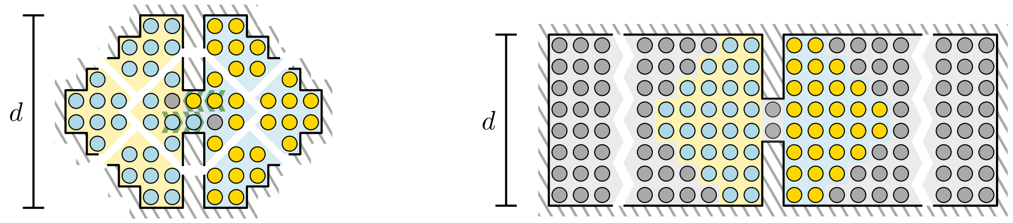
Proof of Theorem 1. Our approach employs methods from Permutation Routing. In this setting, the task is to transform two different vertex labelings of a graph into one another by exchanging labels between adjacent vertices in parallel [5]. A solution (or routing sequence) consists of a series of matchings, i.e., sets of independent edges, along which tokens are exchanged. Such a routing sequence of length at most $3n$ can be computed in almost linear time if the underlying graph is a tree [5]. Thus, we consider an arbitrary spanning tree of a polyomino P's dual graph and compute such a routing sequence. Due to Lemma 4, each parallel swap operation in the sequence can be realized by a schedule of makespan $\mathcal{O}(1)$. We conclude that the schedule derived from the routing sequence has makespan $\mathcal{O}(n)$. ◀

3 The impact of the domain on the achievable makespan

Previous work has demonstrated that it is possible to achieve constant stretch for labeled agents in a rectangular domain. However, in the presence of a non-convex boundary, such stretch factors may not be achievable. We present the following worst-case bound.

▶ **Proposition 5.** *For any $d \geq 5$, there exist instances of diameter d in universally reconfigurable polyominoes, such that all applicable schedules have makespan $\Omega(d^2)$.*

Proof. We illustrate a class of such instances in Figure 4. In this class, agents located on different sides of a narrow passage must trade places. Theorem 1 tells us that these polyominoes are universally reconfigurable; however, any movement between the regions pass through the narrow passage at the center, limiting the number of agents exchanged between them to 2 per transformation. As the number of agents scales quadratically with d, any schedule for this class of instances requires a makespan of $\Omega(d^2)$. ◀



(a) A polyomino with narrow passage. **(b)** This class can be extended such that $n > d^2$.

Figure 4 We illustrate a class of instances which require $\Omega(d^2)$ transformations.

For a more refined characterization of features that affect the achievable makespan and to formulate a precise lower bound, we introduce the following shape parameter for polyominoes.

Bottleneck. We say that the *bottleneck* of a polyomino P is the largest integer $\zeta(P)$, such that there is no non-trivial cut through P of length less than $\zeta(P)$. This means no interior "shortcut" of length less than $\zeta(P)$ exists between any two points on the boundary of P. A *bottleneck cut* through P is therefore a non-trivial cut of length $\zeta(P)$.

We now further refine the lower bound presented in Proposition 5, as follows.

▶ **Proposition 6.** *For any $d \geq 4$ and $z \in [2, d]$, there exists a universally reconfigurable polyomino P with $\zeta(P) = z$ that has instances of diameter d, for which any applicable schedule has a makespan of $\Omega(d^2/\zeta(P))$, i.e., a stretch factor of $\Omega(d/\zeta(P))$.*

Proof. We formulate a generalized version of the instances from Proposition 5. By scaling the boundary of the polyomino between the two regions by an arbitrary amount less or equal to d, we can create universally reconfigurable polyominoes with the targeted bottleneck value.

The movement between the two regions must then still be realized over the narrow grey region, limiting the number of robots exchanged between them to $\mathcal{O}(\zeta(P))$ per transformation. As the number of robots that need to traverse the bottleneck cut scales quadratically with d, any applicable schedule for this class of instances requires a makespan of $\Omega(d^2/\zeta(P))$. ◀

To further refine our understanding of the domain's impact on achievable makespans, we now characterize the size of widest passages, i.e., best case maneuverability. To this end, we consider the maximum shortest distance to the boundary within the given domain, its *depth*.

Depth. We say that the *depth* of a polyomino P is the smallest integer $\mu(P)$, such that every cell in P has geodesic distance at most $\mu(P)$ to the boundary of P.

The depth and bottleneck of a polyomino are very closely related, with depth implying a bound on the bottleneck of any (sub-)polyomino such that $\zeta(P') \leq 2\mu(P)$ for any $P' \subseteq P$. We take particular notice of the following property of depth.

▶ **Lemma 7** ($\star$). *From any cell in a polyomino P, the maximal geodesic distance to a non-trivial geodesic cut of length at most $2\mu(P)$ is also at most $2\mu(P)$.*

4 Bounded makespan for narrow instances

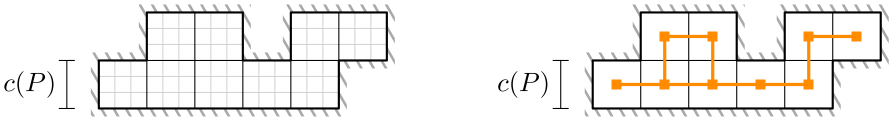
In this section, we consider algorithms for bounded makespan in specific families of instances. Our central result is an approach for asymptotically worst-case optimal stretch in *narrow* instances, which we define as follows. An instance of diameter d in a polyomino P is *narrow*, if and only if $\pi \cdot d \geq \mu(P)$ for some constant $\pi \in \mathbb{N}$, i.e., $\mu(P) \in \mathcal{O}(d)$. Intuitively, these correspond to instances of large diameter relative to the domain's depth.

▶ **Theorem 8** (⋆). *Given an instance of diameter d in a universally reconfigurable polyomino P, we can efficiently compute an applicable schedule of makespan $\mathcal{O}((d+\mu(P))^2/\zeta(P))$. This is asymptotically worst-case optimal for narrow instances.*

As our proof is fairly involved, we proceed with the special case of *scaled polyominoes* in Section 4.1, which we extend to arbitrary polyominoes of limited depth in the subsequent Section 4.2. In each section, we first establish bounds on the makespan relative to a polyomino's area and the corresponding shape parameter.

4.1 Bounded makespan and stretch based on scale

We now investigate *scaled polyominoes*, which we define as follows.



(a) A 3-scaled polyomino P and its tiles. **(b)** The tile dual graph of P.

Figure 5 An illustration of a scaled polyomino P, its tiles, and their corresponding dual graph.

Scaled polyomino. For any $c \in \mathbb{N}$, we say that a polyomino P is c-scaled exactly if it is composed of $c \times c$ squares that are aligned with a corresponding $c \times c$ integer grid. We call these grid-aligned squares *tiles*, which have a *dual graph* analogous to that of a polyomino. Finally, the *scale* of a polyomino P is the largest integer $c(P)$ such that P is c-scaled. This additionally represents a very natural lower bound on the bottleneck, $\zeta(P) \geq c(P)$.

▶ **Proposition 9.** *For any two configurations of a polyomino P with area n and $c(P) \geq 3$, we can compute an applicable schedule of makespan $\mathcal{O}(n/c(P))$ in polynomial time.*

Proof. We model our problem as an instance of Permutation Routing, taking note of two significant results regarding the routing number of specific graph classes. Recall that the routing number $rt(G)$ of a specific graph G refers to the maximum number of necessary routing operations to transform one labeling of G into another. For the complete graph K_n with n vertices, it was shown by Alon, Chung, and Graham [5] that $rt(K_n) = 2$. Furthermore, we make use of a result by Banerjee and Richards [7] which states that for an h-connected graph G and any connected h-vertex induced subgraph G_h of G, the routing number $rt(G)$ is in $\mathcal{O}(rt(G_h) \cdot n/h)$. They also describe an algorithm that determines a routing sequence that matches this bound.

Given a polyomino P and two configurations $C_1, C_2 \in \mathcal{C}(P)$, our goal is to define a secondary graph over the vertices of the dual graph $G(P) = (V, E)$ such that a routing sequence over this graph can be transformed into a schedule $C_1 \rightrightarrows C_2$ of makespan $\mathcal{O}(n/c(P))$.

We define $G_c = (V, E_c)$ such that $\{u, v\} \in E_c$ exactly if the cells u and v are located in the same $c(P) \times c(P)$ tile, or two adjacent tiles. As a result, the cells of each tile in P form a clique, i.e., their induced subgraph is isomorphic to $K_{c(P)^2}$. Furthermore, the cliques of cells in any two adjacent tiles are connected by a set of complete bipartite edges, so they also form a clique. Hence, G_c is h-connected for $h \geq c(P)^2 - 1$ and contains $n/c(P)^2$ cliques of order at least $c(P)^2$. Due to Banerjee and Richards [7], we conclude that $rt(G_c)$ is in $\mathcal{O}(rt(K_{c^2}) \cdot n/h) = \mathcal{O}(n/c(P)^2)$ and can therefore compute a sequence of $\mathcal{O}(n/c(P)^2)$ matchings to route between any two labelings of $G(P)$, which correspond to configurations of P.

It remains to argue that we can realize the swaps induced by any matching in G_c by means of $\mathcal{O}(c(P))$ transformations. All pairwise swaps between cells within the same tile can be realized by applying RotateSort to all tiles in parallel, taking $\mathcal{O}(c(P))$ transformations.

We therefore turn our attention to swaps between adjacent tiles. Observe that the dual graph of the tiles of P is a minor of G_c; contracting the vertices in each of the tile-cliques defined above will give us a corresponding grid graph. Swaps between adjacent tiles can therefore be realized in four phases by covering this grid graph by matchings, and applying RotateSort to the union of matched tile pairs in parallel, again taking $\mathcal{O}(c(P))$ transformations. A cover by four matchings can be determined by first splitting the edges of the dual graph into two sets of horizontal and vertical edges, respectively. Each of these edge sets then induces a collection of paths in the tiling's dual graph, and can therefore be covered by two matchings.

We conclude that constantly many phases of parallel applications of RotateSort suffice to realize any matching in G_c. As $\mathcal{O}(n/c(P)^2)$ matchings can route between any two configurations of P, we conclude that this method yields schedules of makespan $\mathcal{O}(n/c(P))$. ◀

We now apply this intermediate result to compute schedules of bounded stretch in narrow instances of scaled polyominoes. Our approach hinges on the ability to divide the instance into subproblems that can be solved in parallel, which corresponds to cutting the polyomino and performing a sequence of preliminary transformations such that each subpolyomino can then be reconfigured locally, obtaining the target configuration.

Domain partitions. A *partition* of a polyomino P corresponds to a set of disjoint subpolyominoes that cover P. We observe that not every polyomino permits a partition into disjoint universally reconfigurable subpolyominoes. However, any subpolyomino of a universally reconfigurable polyomino P can be made universally reconfigurable by including cells of geodesic distance at most 2 in P.

To efficiently determine such a partition, we employ breadth-first search as follows.

Breadth-first search. For any polyomino Q, let $\texttt{BFS}(Q, v, r)$ refer to the subpolyomino of Q that contains all cells reachable from some cell v in Q by geodesic paths of length at most r. Further, let $\overline{\texttt{BFS}}(Q, v, r)$ refer to the set of connected components of $Q \setminus \texttt{BFS}(Q, v, r)$. We define the *wavefront* of $\texttt{BFS}(Q, v, r)$ as the set of cuts through Q that define the components of $\overline{\texttt{BFS}}(Q, v, r)$. Each connected component (cut) of the wavefront is called a *wavelet.*

▶ **Lemma 10** ($\star$). *For any polyomino Q, the wavefront of $\texttt{BFS}(Q, v, r)$ consists of wavelets of length $\mathcal{O}(\mu(Q))$ each, i.e., the wavelet length is independent of the search radius r.*

Having established all necessary tools, we prove the following statement.

▶ **Proposition 11.** *Given an instance of diameter d in a polyomino P with $c(P) \geq 3$, we can efficiently compute an applicable schedule with makespan $\mathcal{O}((d+\mu(P))^2/c(P))$. This is asymptotically worst-case optimal for narrow instances.*

Proof. We consider an instance of diameter d in a simple polyomino P. We proceed in three phases, which we briefly outline before giving an in-depth description of each.

(I) We partition P into $c(P)$-scaled *patches* of area $\mathcal{O}(d^2)$, using non-trivial cuts of bounded length such that the partition's dual graph is a rooted tree T.

(II) We combine parent/child patches according to T into regions with $c(P)$ scale, allowing us to apply Proposition 9 to reorder them in $\mathcal{O}((d+\mu(P))^2/c(P))$.

(III) Finally, we exploit these combined regions to place all agents at their destination.

Phase (I). A step-by-step illustration of Phase (I) can be found in Figure 6. For this phase, we consider the polyomino P' induced by the tile dual graph of P, recall Figure 5b. This scales the shape parameters and geodesic distance by $1/c(P)$:

$$c(P') = {}^{c(P)}/_{c(P)} = 1, \qquad \mu(P') \approxeq {}^{\mu(P)}/_{c(P)}, \qquad \zeta(P') \approxeq {}^{\zeta(P)}/_{c(P)}.$$

Let $\delta = {}^{3d}/_{c(P)}$. We subdivide P' using a recursive breadth-first-search approach and argue by induction. Given a boundary cell v_0 in P', we determine a patch $P'_0 \subseteq P'$ based on $\texttt{BFS}(P', v_0, \delta) \subseteq P'$. We say that the components of $\overline{\texttt{BFS}}(P', v_0, \delta)$ are either *small* or *large*; a component R is small exactly if $R \subset \texttt{BFS}(P', v_0, 2\delta)$, and large otherwise, see Figure 6b.

We define P'_0 as the union of the initial BFS and the small components of its complement, meaning that for $\overline{\texttt{BFS}}(P', v_0, \delta)$ with large components $R_1, \ldots, R_\ell$, P'_0 takes the shape

$$P'_0 \coloneqq \quad P' \setminus (R_1 \cup \ldots \cup R_\ell).$$

Due to Lemma 10, the cut Γ_i that separates a component R_i of $\overline{\texttt{BFS}}(P', v_0, \delta)$ from P'_0 has length $\mathcal{O}(\mu(P'))$. By definition, the geodesic distance from each cell in R_i to v_0 is at least δ.

We now iteratively subdivide each component of $P' \setminus P'_0$ by simply increasing the maximal depth of our BFS from v_0 by another δ units and again considering large and small components of the corresponding subdivision separately, as illustrated in Figures 6c and 6d.

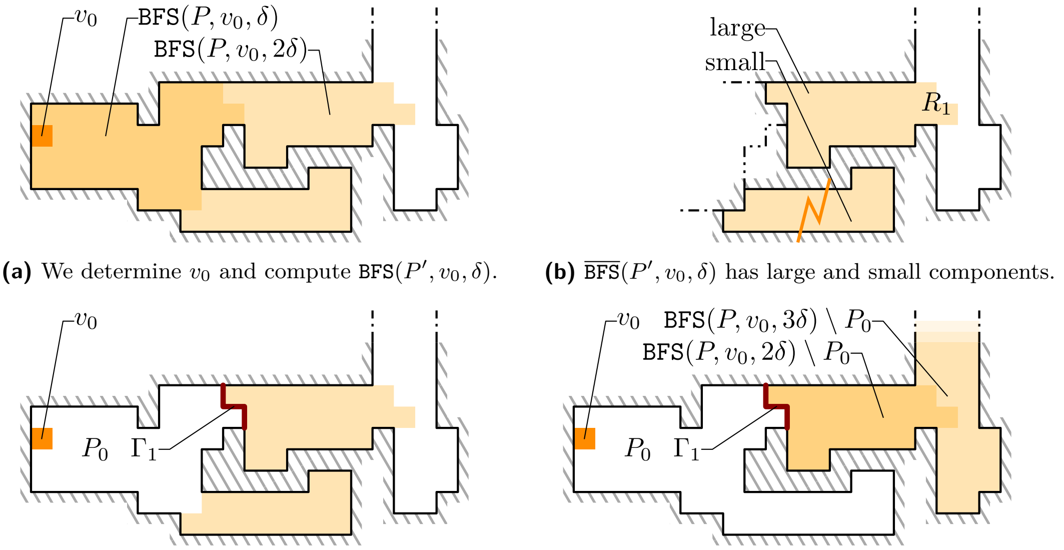
To obtain a partition of P, we map each patch P'_i to the tiles in P that its cells correspond to. Since P is a simple polyomino, the dual graph of our patches forms a tree T rooted at P_0.

Consider any patch P'_i and recall that, due to Lemma 10, all cuts induced by BFS have individual length $\mathcal{O}(\mu(P'))$. Tracing along the boundaries of tiles, we conclude that the corresponding cuts in P have individual length $\mathcal{O}(\mu(P')c(P))) = \mathcal{O}(\mu(P))$. Due to triangle inequality, it follows that any two cells in each patch P_i have geodesic distance $\mathcal{O}(d + \mu(P))$. From this, we conclude that the area of P_i is bounded by $\mathcal{O}((d + \mu(P))^2)$. It directly follows that for any patch P_j with hop distance $k \in \mathbb{N}^+$ to P_i in T, the geodesic distance between two cells in P_i and P_j is bounded by $\mathcal{O}(k(d + \mu(P)))$. The union of patches in a subtree T' of T with height k therefore has area $\mathcal{O}((k(d + \mu(P)))^2)$.

Phase (II). We use this partition of P into patches to subdivide the instance into disjoint tasks that can be solved in parallel; recall that our target makespan is $\mathcal{O}({}^{(d+\mu(P))^2}/_{\zeta(P)})$. The patches are spatially disjoint and all have scale at least $c(P)$, as well as area $\mathcal{O}((d + \mu(P))^2)$. Proposition 9 therefore implies that the patches can be locally reconfigured in parallel, by schedules of makespan $\mathcal{O}({}^{(d+\mu(P))^2}/_{\zeta(P)})$. In order to solve the original instance, it therefore remains to make each patch a subproblem that can be solved independently.

We argue that we can efficiently move robots into their target patches. In Phase (I), we gave an upper bound of $\mathcal{O}(k(d + \mu(P)))$ on the geodesic distance between cells in patches that have hop distance at most $k \in \mathbb{N}^+$ in T. We now provide a lower bound: The geodesic distance between cells in patches that are not in a parent-child or sibling relationship in T is at least d, as the distance between cells in any patch and its "grandparent" patch according to T is at least d by construction, see Phase (I).

It follows that the target cell of each robot is either in the same patch as its initial cell, or in a parent or sibling thereof. To realize the movement of agents between patches, we thus simply form the spatial union F_i of each patch P_i and its children according to T. Each of the resulting subpolyominoes F_i has area $\mathcal{O}((d + \mu(P))^2)$. As T is bipartite, we can split them into two sets $\mathcal{F}_A$ and $\mathcal{F}_B$, each comprised of pairwise spatially disjoint subpolyominoes.


v_0
$\mathtt{BFS}(P, v_0, \delta)$
$\mathtt{BFS}(P, v_0, 2\delta)$
large
small
R_1
v_0
P_0
Γ_1
v_0
$\mathtt{BFS}(P, v_0, 3\delta) \setminus P_0$
$\mathtt{BFS}(P, v_0, 2\delta) \setminus P_0$
P_0
Γ_1

(a) We determine v_0 and compute $\mathtt{BFS}(P', v_0, \delta)$.

(b) $\overline{\mathtt{BFS}}(P', v_0, \delta)$ has large and small components.

(c) The patch P'_0 is the union of $\mathtt{BFS}(P', v_0, \delta)$ and the small components of $\overline{\mathtt{BFS}}(P', v_0, \delta)$.

(d) We continue the breadth-first-search in $P \setminus P_0$.

Figure 6 Phase (I): We divide P' into patches of area $\mathcal{O}((\delta + \mu(P'))^2)$.

Phase (III). It remains to show that we can efficiently exchange agents between patches. Note that, as P is simple, the number of agents that need to cross any cut in either direction is equal to that for the opposite direction.

By construction, every pair of patches that needs to exchange agents between one another is fully contained in some $F_i \in (\mathcal{F}_A \cup \mathcal{F}_B)$. We proceed in three iterations: By applying Proposition 9 to each of the patches in $\mathcal{F}_A$ in parallel, we swap agents across cuts by swapping them with agents moving in the opposite direction. We repeat this process for $\mathcal{F}_B$ and finally perform a parallel reconfiguration of the individual patches, which allows us to place every robot in its target cell. Each iteration takes $\mathcal{O}(^{(d+\mu(P))^2}/_{\zeta(P)})$ transformations. ◀

4.2 Bounded makespan and stretch based on bottleneck

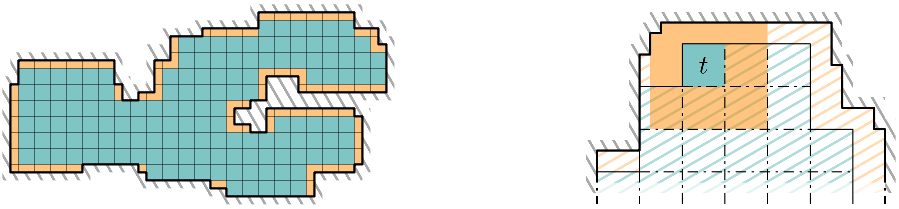
Finally, this section concerns itself with the transfer of results from Section 4.1 to arbitrary polyominoes. As this requires significantly more intricate local mechanisms, we only provide a high-level description of the necessary tools and modifications, and refer to the full version.

Skeleton. A *skeleton* of a polyomino P is a connected, λ-scaled subpolyomino $S \subseteq P$ with $\lambda = \lfloor ^{\zeta(P)}/4 \rfloor$, as illustrated in Figure 7a. Such a skeleton can easily be determined as the union of all $\lambda \times \lambda$ squares in P that are aligned with the same $\lambda \times \lambda$ integer grid.

Watershed. The *watershed* of a skeleton tile t corresponds to the union of all $2\lambda \times 2\lambda$ squares in P that fully contain t, see Figure 7b. We will show that at least one such square always exists, and their union forms a convex polyomino with bottleneck at least λ.

To swap agents from $P \setminus S$ into the skeleton S, we exploit the watersheds of its tiles. We first prove the existence of a skeleton $S \subseteq P$ as above, and that its watersheds fully cover P. This allows us to apply techniques from Section 4.1 to arbitrary polyominoes.

▶ **Lemma 12** (⋆). *For any polyomino P with $\zeta(P) \geq 8$, we can identify a skeleton $S \subseteq P$ in polynomial time. The skeleton S is λ-scaled for $\lambda = \lfloor ^{\zeta(P)}/4 \rfloor$, and every $2\lambda \times 2\lambda$ square inside P contains at least one of its scaled tiles.*



(a) A polyomino P and its skeleton S (cyan). **(b)** An illustration of a skeleton tile t's watershed.

Figure 7 The central tools used in our proof of Theorem 15.

We make use of the following result obtained by Alpert et al. [6] for the routing number of convex grid graphs, where $w(P)$ and $h(P)$ refer to the width and height of P, respectively.

▶ **Theorem 13** (Alpert et al. [6]). *Let P be a connected convex grid piece. Then the routing number of P satisfies the bound $rt(P) \leq C(w(P) + h(P))$ for some universal constant C.*

Finally, we demonstrate a method for efficient reconfiguration of an arbitrary skeleton tile's watershed in order to swap robots into and out of the skeleton.

▶ **Lemma 14.** *Given two configurations of a skeleton tile's watershed in a universally reconfigurable polyomino, we can efficiently compute an applicable schedule of makespan $\mathcal{O}(\lambda)$.*

Proof. Consider a $\lambda \times \lambda$ skeleton tile $t \subset P$ of a universally reconfigurable polyomino P, and let H refer to its watershed. Recall that we assume $\zeta(P) \geq 8$, so $\lambda = \lfloor \zeta(P)/4 \rfloor \geq 2$. As H is the union of all $2\lambda \times 2\lambda$ squares in P that contain t, it is thus universally reconfigurable and orthoconvex. We apply Theorem 13: Alpert et al. [6] presented a constructive proof in the form of an algorithm, which we can use to compute a routing sequence of length $\mathcal{O}(w(H)+h(H)) = \mathcal{O}(\lambda)$ between any two configurations of H, based on its dual graph. Such a routing sequence corresponds to a series of matchings in the dual graph of H that exchange tokens of adjacent vertices. Sequentially realizing these matchings by swapping adjacent agents as outlined in Lemma 4, we can arbitrarily reorder H in $\mathcal{O}(\lambda)$ transformations. ◀

This provides us with all necessary tools to prove the following generalization of Proposition 9, which, in turn, is a central tool for our proof of Theorem 8.

▶ **Theorem 15** ($\star$). *For any two configurations of a universally reconfigurable polyomino P of area n, we can compute an applicable schedule of makespan $\mathcal{O}(n/\zeta(P))$ in polynomial time.*

▶ **Theorem 8** ($\star$). *Given an instance of diameter d in a universally reconfigurable polyomino P, we can efficiently compute an applicable schedule of makespan $\mathcal{O}((d+\mu(P))^2/\zeta(P))$. This is asymptotically worst-case optimal for narrow instances.*

Proof sketch. Assuming that $c(P) < \zeta(P)$, our approach consists of three phases:

(I) We compute a skeleton $S \subset P$ which we split into patches S_i according to Phase (I) of Proposition 11. To each of these patches, we add cells of its skeleton tile's watersheds. This partitions P into patches P_i, each with a skeleton patch $S_i \subset S$ such that $S_i \subseteq P_i$.

(II) We use the rooted dual tree T of the skeleton patches S_i and, for each patch S_i with children $S_\ell, \dots, S_{\ell+k}$ according to T, we combine the patches $P_i, P_\ell, \dots, P_{\ell+k}$ to a (not necessarily connected) region F_i that can be reordered in $\mathcal{O}((d+\mu(P))^2/\zeta(P))$.

(III) Finally, we exploit these combined regions to place all agents at their destination.

These act analogously to Phases (I) – (III) of Proposition 11. ◀

5 Conclusions and future work

We provide a number of novel contributions for Multi-Agent Path Finding in simple polyominoes. We establish a characterization for the existence of reconfiguration schedules, based on different shape parameters of the bounding polyomino. Furthermore, we establish algorithmic methods that achieve worst-case optimal stretch for any instance in which the polyomino's bottleneck does not exceed the instance's diameter by more than a constant factor. There are a variety of directions in which these insights should give rise to further generalizations and applications.

Non-simple polyominoes. Our results regarding universal reconfigurability are directly applicable to non-simple polyominoes. As noted in Section 2, the geometric characterization for simple polyominoes is formed as a special case based on the dual graph of a polyomino.

For any non-simple polyomino that is either 2-scaled or 2-square-connected, Theorem 1 and Proposition 9 are also directly applicable. The same is not true for Theorem 15, as our definition of the bottleneck based on cuts does not work in this case. However, with a separate definition that accounts for the minimal distance between inner and outer boundaries, Theorem 15 may be applicable.

Permutation routing. Our results can be generalized to solid grid graph routing, which is a generalization of the findings of Alpert et al. [6]. We provided results regarding bounded stretch for this setting, therefore tackling a special case of their Open Question 2.

Further questions. Our work is orthogonal to that of Demaine et al. [13, 14]: Their setting considered domains of large depth in conjunction with large bottleneck, i.e., the case that $\mu(P) \in \Omega(d)$ and $\zeta(P) \in \Omega(d)$. We establish asymptotically worst-case optimal results for narrow domains, which implies that $\mu(P) \in \mathcal{O}(d)$ and $\zeta(P) \in \mathcal{O}(d)$. In particular, instances where $\zeta(P) \in o(d)$ while $\mu(P) \in \omega(d)$, i.e., instances in which the gap between bottleneck and depth is unbounded relative to d, remain a challenge even for simple domains. We conjecture that this question for simple domains is equivalent to that of bounded stretch for non-simple domains with limited depth; considering an instance of large depth, we can create an analogous non-simple instance that features regularly distributed, small holes based on some grid graph. This may motivate research into the special case of instances in which the diameter is less or equal to the circumference of the smallest hole in the domain.

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.

2 Pankaj K. Agarwal, Tzvika Geft, Dan Halperin, and Erin Taylor. Multi-robot motion planning for unit discs with revolving areas. *Computational Geometry: Theory and Applications*, 114:102019, 2023. doi:10.1016/J.COMGEO.2023.102019.

3 Pankaj K. Agarwal, Dan Halperin, Micha Sharir, and Alex Steiger. Near-optimal min-sum motion planning for two square robots in a polygonal environment. In *Symposium on Discrete Algorithms (SODA)*, pages 4942–4962, 2024. doi:10.1137/1.9781611977912.176.

4 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. In *European Symposium on Algorithms (ESA)*, pages 3:1–3:15, 2022. doi:10.4230/LIPICS.ESA.2022.3.

5 Noga Alon, Fan R. K. Chung, and Ronald L. Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7(3):513–530, 1994. doi:10.1137/S0895480192236628.

6 H. Alpert, R. Barnes, S. Bell, A. Mauro, N. Nevo, N. Tucker, and H. Yang. Routing by matching on convex pieces of grid graphs. *Computational Geometry*, 104:101862, 2022. doi:10.1016/j.comgeo.2022.101862.

7 Indranil Banerjee and Dana Richards. New results on routing via matchings on graphs. In *Fundamentals of Computation Theory (FCT)*, pages 69–81, 2017. doi:10.1007/978-3-662-55751-8_7.

8 Bahareh Banyassady, Mark de Berg, Karl Bringmann, Kevin Buchin, Henning Fernau, Dan Halperin, Irina Kostitsyna, Yoshio Okamoto, and Stijn Slot. Unlabeled multi-robot motion planning with tighter separation bounds. In *Symposium on Computational Geometry (SoCG)*, pages 12:1–12:16, 2022. doi:10.4230/LIPICS.SOCG.2022.12.

9 Marc Baumslag and Fred S. Annexstein. A unified framework for off-line permutation routing in parallel networks. *Mathematical Systems Theory*, 24(4):233–251, 1991. doi:10.1007/BF02090401.

10 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated motion planning: The video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. doi:10.4230/LIPICS.SOCG.2018.74.

11 Soon-Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018. doi:10.1109/TRO.2018.2857475.

12 Loïc Crombez, Guilherme Dias da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso. Shadoks approach to low-makespan coordinated motion planning. *ACM Journal of Experimental Algorithmics*, 27:3.2:1–3.2:17, 2022. doi:10.1145/3524133.

13 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. In *Symposium on Computational Geometry (SoCG)*, pages 29:1–29:15, 2018. doi:10.4230/LIPICS.SOCG.2018.29.

14 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.

15 Eduard Eiben, Robert Ganian, and Iyad Kanj. The parameterized complexity of coordinated motion planning. In *Symposium on Computational Geometry (SoCG)*, pages 28:1–28:16, 2023. doi:10.4230/LIPICS.SOCG.2023.28.

16 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. *Autonomous Agents and Multi-Agent Systems*, 37(2):43, 2023. doi:10.1007/S10458-023-09626-5.

17 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG:SHOP challenge 2021. *ACM Journal of Experimental Algorithmics*, 27:3.1:1–3.1:12, 2022. doi:10.1145/3532773.

18 Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Jonas Neutzner, Christian Rieck, and Christian Scheffer. Coordinated motion planning: Multi-agent path finding in a densely packed, bounded domain, 2024. doi:10.48550/arXiv.2409.06486.

19 Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots. *Autonomous Agents and Multi-Agent Systems*, 38(2):39, 2024. doi:10.1007/S10458-024-09668-3.

20 Lenwood S. Heath and John Paul C. Vergara. Sorting by short swaps. *Journal of Computational Biology*, 10(5):775–789, 2003. doi:10.1089/106652703322539097.

21 John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the warehouseman's problem. *International Journal of Robotics Research*, 3(4):76–88, 1984. doi:10.1177/027836498400300405.

22 John E. Hopcroft and Gordon T. Wilfong. Reducing multiple object motion planning to graph searching. *SIAM Journal on Computing*, 15(3):768–785, 1986. doi:10.1137/0215055.

23 Jun Kawahara, Toshiki Saitoh, and Ryo Yoshinaka. The time complexity of permutation routing via matching, token swapping and a variant. *Journal of Graph Algorithms and Applications*, 23(1):29–70, 2019. doi:10.7155/jgaa.00483.

24 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-opt. *ACM Journal of Experimental Algorithmics*, 27:3.4:1–3.4:9, 2022. doi:10.1145/3524134.

25 John M. Marberg and Eli Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, 3:561–572, 1988. doi:10.1007/BF01762132.

26 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *European Symposium on Algorithms (ESA)*, pages 66:1–66:15, 2016. doi:10.4230/LIPICS.ESA.2016.66.

27 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014. doi:10.1126/science.1254295.

28 Erol Şahin and Alan F. T. Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2-4):69–72, 2008. doi:10.1007/s11721-008-0020-6.

29 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.

30 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *International Journal of Robotics Research*, 35(14):1750–1759, 2016. doi:10.1177/0278364916672311.

31 Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Symposium on Combinatorial Search (SOCS)*, pages 151–158, 2019. doi:10.1609/SOCS.V10I1.18510.

32 Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–19, 2008. doi:10.1609/aimag.v29i1.2082.

33 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015. doi:10.1016/J.TCS.2015.01.052.

34 Hyeyun Yang and Antoine Vigneron. Coordinated path planning through local search and simulated annealing. *ACM Journal of Experimental Algorithmics*, 27:3.3:1–3.3:14, 2022. doi:10.1145/3531224.

35 Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 729–746, 2015. doi:10.1007/978-3-319-16595-0_42.

On the Complexity of Establishing Hereditary Graph Properties via Vertex Splitting

Alexander Firbas
TU Wien, Austria

Manuel Sorge
TU Wien, Austria

Abstract

Vertex splitting is a graph operation that replaces a vertex v with two nonadjacent new vertices u, w and makes each neighbor of v adjacent with one or both of u or w. Vertex splitting has been used in contexts from circuit design to statistical analysis. In this work, we generalize from specific vertex-splitting problems and systematically explore the computational complexity of achieving a given graph property Π by a limited number of vertex splits, formalized as the problem Π Vertex Splitting (Π-VS). We focus on hereditary graph properties and contribute four groups of results: First, we classify the classical complexity of Π-VS for graph properties characterized by forbidden subgraphs of order at most 3. Second, we provide a framework that allows one to show NP-completeness whenever one can construct a combination of a forbidden subgraph and prescribed vertex splits that satisfy certain conditions. Using this framework we show NP-completeness when Π is characterized by sufficiently well-connected forbidden subgraphs. In particular, we show that F-Free-VS is NP-complete for each biconnected graph F. Third, we study infinite families of forbidden subgraphs, obtaining NP-completeness for Bipartite-VS and Perfect-VS, contrasting the known result that Π-VS is in P if Π is the set of all cycles. Finally, we contribute to the study of the parameterized complexity of Π-VS with respect to the number of allowed splits. We show para-NP-hardness for K_3-Free-VS and derive an XP-algorithm when each vertex is only allowed to be split at most once, showing that the ability to split a vertex more than once is a key driver of the problems' complexity.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases NP-completeness, polynomial-time solvability, graph theory, graph transformation, graph modification

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.30

Related Version *Full Version*: `https://arxiv.org/abs/2401.16296` [15]

Funding Alexander Firbas acknowledges support from the Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029]. Manuel Sorge acknowledges partial support from the Alexander von Humboldt Foundation.

1 Introduction

Vertex splitting is the graph operation in which we take a vertex v, remove it from the graph, add two *descendants* v_1, v_2 of v, and make each former neighbor of v adjacent with v_1, v_2, or both. Vertex splitting has been used in circuit design [28, 31], the visualization of nonplanar graphs in a planar way [2, 4, 11, 12, 24, 30], improving force-based graph layouts [10], in graph clustering with overlaps [1, 3, 5, 14], in statistics [9, 20] (see [14]), in subgraph counting [18, 32], and variants of vertex splitting in which we may make the copies adjacent play roles in graph theory [25, 29], in particular in Fleischner's Splitting Lemma [16] and in Tutte's theorem relating wheels and general three-connected graphs [33]. Such a variant of vertex splitting can also be thought of as an inverse operation of vertex contraction, which is an underlying operation of the graph parameters twinwidth (see, e.g., [6]) and fusion-width [7, 17].

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 30; pp. 30:1–30:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In some of the above applications, we are given a graph and want to establish a graph property by splitting the least number of times: In circuit design, we aim to bound the longest path length [28, 31], when visualizing non-planar graphs we aim to establish planarity [2, 11, 12, 30] or pathwidth one [4], and in statistics and when clustering with overlaps we want to obtain a cluster graph (a disjoint union of cliques) [1, 3, 9, 14, 20].

This motivates generalizing these problems by letting Π be any graph property (a family of graphs) and studying the problem Π Vertex Splitting (Π-VS): Given a graph G and an integer k, is it possible to apply at most k vertex split operations to G to obtain a graph in Π? The above-mentioned graph properties are closed under taking induced subgraphs and thus we mainly focus on this case. For graph operations different from vertex splitting the complexity of establishing graph properties Π is well studied, such as for deleting vertices (e.g., [26, 27]), adding or deleting edges (see the recent survey [8]), or edge contractions (e.g., [19, 21–23]). In this work, we aim to start this direction for vertex splitting, that is, how can we characterize for which graph properties Π-VS is tractable? Our main focus here is the classical complexity, that is, NP-hardness vs. polynomial-time solvability, but we also touch on the parameterized complexity with respect to the number of allowed splits.

Our results are as follows. Each graph property Π that is closed under taking induced subgraphs is characterized by a family $\mathcal{F}$ of *forbidden induced subgraphs*. We also write Π as $\text{Free}_{\prec}(\mathcal{F})$. It is thus natural to begin by considering small forbidden subgraphs. We classify for each family $\mathcal{F}$ that contains graphs of order at most 3 whether $\text{Free}_{\prec}(\mathcal{F})$-VS is polynomial-time solvable or NP-complete. Indeed, it is NP-complete precisely if $\mathcal{F}$ contains only the path P_3 on three vertices or a triangle K_3:

▶ **Theorem 1.1 (★).** *Let $\mathcal{F}$ be a set of graphs containing graphs of at most three vertices each. Then, $\text{Free}_{\prec}(\mathcal{F})$-VS is NP-complete if $\mathcal{F} = \{P_3\}$ or $\mathcal{F} = \{K_3\}$ and is in P otherwise.*

The polynomial-time results use a plethora of different approaches and also extend to Threshold-VS and Split-VS. The NP-hardness for $\Pi = \text{Free}_{\prec}(\{K_3\})$ can be shown using a reduction from the Vertex Cover problem. In this reduction, we replace each edge of a graph by a K_3. It is then not hard to show a correspondence between splitting a set of at most k vertices to destroy all induced K_3 and a vertex cover of size at most k for the original graph. Together with our results below, we also obtain NP-completeness for each connected forbidden subgraph F with four vertices except for P_4s and claws $K_{1,3}$, for which the complexity remains open.

Second, the hardness construction for K_3-free graphs indicates that high connectivity in forbidden subgraphs makes Π-VS hard and thus we explored this direction further. As the naïve approach breaks down in the general setting, we reduce from a special variant of Vertex Cover and develop a framework for showing NP-hardness of Π-VS whenever one can use forbidden induced subgraphs to construct certain splitting configurations (Section 3). That is, a graph H together with a recipe specifying distinguished vertices that will be connected to the outside of H and how to split them. Essentially, if one can provide a splitting configuration that avoids introducing new forbidden subgraphs and that decreases the connectivity to the outside well enough, then we can use such a configuration to give a hardness construction. We then provide ways to obtain such splitting configurations, allowing us to show the following hardness results, where we write $\text{Free}_{\subseteq}(\mathcal{F})$ to exclude the graphs in $\mathcal{F}$ as subgraphs, rather than induced subgraphs:

▶ **Theorem 1.2 (★).** *Let $\mathcal{F}$ be a family of graphs.*

1. *If $\mathcal{F}$ consists of a single biconnected graph, then $\text{Free}_{\prec}(\mathcal{F})$-VS and $\text{Free}_{\subseteq}(\mathcal{F})$-VS are NP-complete.*

2. *If all graphs in $\mathcal{F}$ are triconnected and the family has bounded diameter, then* $\text{Free}_{\prec}(\mathcal{F})$*-VS and* $\text{Free}_{\subseteq}(\mathcal{F})$*-VS are* NP*-hard.*
3. *If all graphs in $\mathcal{F}$ are 4-connected, then* $\text{Free}_{\prec}(\mathcal{F})$*-VS and* $\text{Free}_{\subseteq}(\mathcal{F})$*-VS are* NP*-hard.*

NP*-completeness in item 2 and 3 holds when* $\text{Free}_{\prec/\subseteq}(\mathcal{F})$ *is decidable in polynomial time.*

Third, the above results do not cover the case where $\mathcal{F}$ is the family of all cycles, and this must be so because Forest-VS is polynomial-time solvable [4, 13]. However, we show that if we forbid only cycles of at most a certain length, or all cycles of odd length, then Π-VS becomes NP-complete again. This hardness extends also to perfect graphs:

▶ **Theorem 1.3** (★). *Bipartite-VS and Perfect-VS are* NP*-complete.*

The hardness construction for Bipartite Vertex Splitting is similar to the one for $\text{Free}_{\prec}(\{K_3\})$-VS mentioned above. The nontrivial part of the proof is, given a vertex cover, how to split vertices such that the resulting graph is two-colorable. By carefully checking all the possible configurations of vertices in the vertex cover and splitting them in the right way, we obtain subconfigurations that are two-colorable and whose colorings can be combined into a two-coloring for the whole graph. The reduction for Perfect Vertex Splitting uses five-vertex cycles instead of K_3's and the correctness additionally uses a degree-based argument to show that the graph after splitting is also odd-anti-hole-free and thereby perfect.

Finally, we contribute to the parameterized complexity of Π-VS with respect to the number k of allowed vertex splits. Previously it was known that Π-VS is fixed-parameter tractable when Π is closed under taking minors [30], when $\Pi = \text{Free}_{\prec}(\{P_3\})$ [13, 14], and when Π consists of graphs of pathwidth one or when Π is MSO_2-definable and of bounded treewidth [4]. In contrast, we observe that $\text{Free}_{\prec}(\{K_3\})$-VS is NP-hard even for $k = 2$:

▶ **Theorem 1.4** (★). $\text{Free}_{\prec}(\{K_3\})$*-VS is* NP*-complete for two splits.*

The idea behind this result is to reduce from 3-Coloring on K_3-free graphs: Barring the technical details, we add a universal vertex u to a graph G, and gadgets to ensure that the vertex u must be split. Then, the constraint that two adjacent vertices v, w in G need to be colored differently translates to the constraint that v and w need to be adjacent to different descendants of u. The crux herein is that one can split a vertex multiple times: In contrast, if we instead can split each vertex at most once, resulting in the problem Shallow Triangle-Free Vertex Splitting, then we obtain an XP algorithm:

▶ **Theorem 1.5** (★). *Shallow Triangle-Free Vertex Splitting, parameterized by the number k of splits, admits an $O(\sqrt{2}^{k^2} \cdot n^{k+3})$-time* XP *algorithm.*

The basic idea is that we can guess which vertices are split and how they are split with respect to each other. Formulating the condition that the guess was correct can then be done using a 2-SAT formula.

Due to space constraints, we only provide details for the dichotomy result for small forbidden subgraphs (Theorem 1.1) and the framework for proving Theorem 1.2. All remaining results are marked with ★ and are proved in the full version of this paper [15].

1.1 Preliminaries

General (Graph) Notation. For a function $f\colon A \to B$, we let $\text{Domain}(f) := A$ and $\text{Range}(f) := \{b \mid \exists a \in A\colon f(a) = b\}$. For a set X, we let $\mathcal{P}(X)$ be its power set. Unless stated otherwise, all graphs are undirected and without parallel edges or self-loops. Let G

be a graph with vertex set $V(G)$ and edge set $E(G)$. We denote the neighborhood of a vertex $v \in V(G)$ by $N_G(v)$. The graph induced by a vertex set $V' \subseteq V(G)$ is written as $G[V']$. For $u, v \in V(G)$, we write uv as a shorthand for $\{u, v\}$, $G - v$ for $G[V(G) \setminus \{v\}]$, $\deg_G(v)$ for $|N_G(v)|$, $d_G(u,v)$ for the length of a shortest path from u to v if there is one and ∞ otherwise, and $\operatorname{diam}(G)$ for the diameter of G, that is, $\max_{u,v \in V(G)} d_G(u,v)$. We denote the complement of G by $\overline{G}$. The graph K_n is the complete graph on n vertices and C_n the cycle graph of n vertices. If a graph G is isomorphic to a graph H, we write $G \simeq H$. The *circumference* of a graph G is the length of a longest cycle of G if G it is not acyclic and zero otherwise. A *k-subdivision* of a graph G is a graph obtained by replacing each of G's edges uv with a path $u, p_1^{uv}, p_2^{uv}, \ldots, p_k^{uv}, v$, where $p_1^{uv}, p_2^{uv}, \ldots, p_k^{uv}$ are new vertices. We mark directed graphs $\vec{G}$ with an arrow. For an arc $uv \in E(\vec{G})$, u is the source vertex and v is the target vertex. All directed graphs are oriented, that is, for each $uv \in E(\vec{G})$, we have $vu \notin E(\vec{G})$. We denote the in-neighborhood by $N^-_{\vec{G}}(\cdot)$ and the out-neighborhood by $N^+_{\vec{G}}(\cdot)$. A directed graph $\vec{G}$ is an *orientation* of G if the underlying undirected graph of $\vec{G}$ is G.

Vertex Splitting. Let G be a graph, $v \in V(G)$, and V_1, V_2 subsets of $N_G(v)$ such that $V_1 \cup V_2 = N_G(v)$. Furthermore, let v_1 and v_2 denote two fresh vertices, that is, $\{v_1, v_2\} \cap V(G) = \varnothing$. Consider the graph G' that is obtained from G by deleting v, and adding v_1 and v_2 such that $N_{G'}(v_1) = V_1$ and $N_{G'}(v_2) = V_2$. Then, we say G' was obtained from G by *splitting* v (via a *vertex split*). If $V_1 \cap V_2 = \varnothing$, we speak of a *disjoint* vertex split, and if either $V_1 = \varnothing$ or $V_2 = \varnothing$, we say the split is *trivial.* Furthermore, we say v was *split into* v_1 and v_2, and call these vertices the *descendants* of v. Conversely, v is called the *ancestor* of v_1 and v_2. Finally, consider an edge v_1w (resp. v_2w) of G'. We say that the edge vw of G was *assigned* to v_1 (resp. v_2) in the split, and call v_1w (resp. v_2w) a *descendant edge* of vw.

A *splitting sequence* of k splits is a sequence of graphs $G_0, G_1, \ldots, G_k$, such that G_{i+1} is obtainable from G_i via a vertex split for $i \in \{0, \ldots, k-1\}$. The notion of descendant vertices (resp. ancestor vertices) is extended in a transitive and reflexive way (that is, a vertex is its own ancester and descendant) to splitting sequences.

Later, the following shorthand notation will be useful: Let H be a graph, $v \in V(H)$, $X_1, X_2 \subseteq N_H(v)$ with $X_1 \cup X_2 = N_H(v)$, and v_1, v_2 two distinct vertices. Further, let H' be the graph obtained by splitting v into v_1 and v_2 while setting $N_{H'}(v_1) = X_1$, $N_{H'}(v_2) = X_2$. Then, we identify H' with the shorthand $\operatorname{Split}(H, v, X_1, X_2, v_1, v_2)$.

Embeddings and Hereditary Graph Properties. For graphs G and H, we write $\operatorname{Emb}_{\prec}(G,H)$ (resp. $\operatorname{Emb}_{\subseteq}(G,H)$) to denote the set of all *induced embeddings* of G in H (resp. *subgraph embeddings*), that is, the set of all injective $f \colon V(G) \to V(H)$ where $\forall uv \in V(G)^2 \colon uv \in E(G) \iff f(u)f(v) \in E(H)$ (resp. $\forall uv \in V(G)^2 \colon uv \in E(G) \implies f(u)f(v) \in E(H)$). In case $\operatorname{Emb}_{\prec}(G,H) \neq \varnothing$ (resp. $\operatorname{Emb}_{\subseteq}(G,H) \neq \varnothing$), we write $G \prec H$ (resp. $G \subseteq H$) and say G is an *induced subgraph* (resp. a *subgraph*) of H.

For a set of graphs $\mathcal{F}$, we write $\operatorname{Free}_{\prec}(\mathcal{F})$ (resp. $\operatorname{Free}_{\subseteq}(\mathcal{F})$) to denote the set of graphs where $G \in \operatorname{Free}_{\prec}(\mathcal{F})$ (resp. $G \in \operatorname{Free}_{\subseteq}(\mathcal{F})$) iff $\operatorname{Emb}_{\prec}(F,G) = \varnothing$ (resp. $\operatorname{Emb}_{\subseteq}(F,G) = \varnothing$) for all $F \in \mathcal{F}$. Set $\mathcal{F}$ is the set of *forbidden induced subgraphs* (resp. *forbidden subgraphs*) that *characterize* the *hereditary (graph) property* $\operatorname{Free}_{\prec}(\mathcal{F})$ (resp. $\operatorname{Free}_{\subseteq}(\mathcal{F})$).

2 Properties Characterized by Small Forbidden Induced Subgraphs

We now give an outline of the characterization of Π-VS for Π characterized by families $\mathcal{F}$ of forbidden induced subgraphs with at most three vertices. The full version of this section is given in the full version of this paper [15]. First, we can make several simple observations: If one of K_0, K_1, K_2, or $\overline{K_2}$ is forbidden and it is present in the input graph, then there is no way to destroy these forbidden subgraphs with vertex splitting and hence we can immediately return a failure symbol. This gives a trivial algorithm if $K_0 \in \mathcal{F}$ or $K_1 \in \mathcal{F}$. Moreover, if $\overline{K_2} \in \mathcal{F}$, then the input graph is a clique or we can return failure. Since splitting introduces a $\overline{K_2}$, instance (G, k) is positive if and only if $(G, 0)$ is positive, which we can check in polynomial time. Similarly, if $K_2 \in \mathcal{F}$, then the input graph is an independent set or we can return failure. Through splitting, we can only introduce more independent vertices and thus (G, k) is positive if and only if $(G, 0)$ is positive.

It follows that we can focus on families $\mathcal{F}$ that contain subgraphs with exactly 3 vertices, that is, $\mathcal{F} \subseteq \{P_3, \overline{P_3}, K_3, \overline{K_3}\}$. If $\mathcal{F}$ contains $\overline{P_3}$ or $\overline{K_3}$ but neither P_3 nor K_3, then we have a similar observation as above: $\overline{P_3}$ and $\overline{K_3}$ cannot be destroyed by vertex splits and thus (G, k) is positive if and only if $(G, 0)$ is, which is checkable in polynomial time.

It thus remains to classify families $\mathcal{F} \subseteq \{P_3, \overline{P_3}, K_3, \overline{K_3}\}$ that contain P_3 or K_3. If $\mathcal{F} = \{P_3\}$ then $\text{Free}_{\prec}(\mathcal{F})$-VS is NP-complete by a result of Firbas et al. [14, Theorem 4.4]. If $\mathcal{F} = \{K_3\}$ then NP-completeness follows from Theorem 1.2 or Theorem 1.4, which we prove below. However, if we combine P_3 and K_3 or if we add $\overline{P_3}$ and/or $\overline{K_3}$ then the problems once again become polynomial-time solvable for subtle and different reasons:

For $\mathcal{F} = \{P_3, K_3\}$ all solution graphs are the union of an independent set and a matching and there is essentially only one minimal sequence of vertex splits. In the case where $\{K_3, \overline{K_3}\} \subseteq \mathcal{F}$ we can apply Ramsey-type arguments to show that an algorithm only needs to check for a constant number of different yes-instances. If $\overline{P_3} \in \mathcal{F}$ we can observe that destroying any P_3 or K_3 necessarily introduces a $\overline{P_3}$, which cannot be removed afterwards.

This takes care of all cases for $\mathcal{F}$ except $\mathcal{F} = \{P_3, \overline{K_3}\}$. For this case we can observe that the graphs resulting from a splitting solution are *cluster graphs*, disjoint unions of cliques, with at most two clusters (cliques). As $\overline{K_3}$ cannot be destroyed by vertex splitting, the input graph may only contain P_3s. Furthermore, P_3s can only be destroyed by splitting their midpoints. It is thus intuitive that the input graph of a yes-instance must consist of two cliques that may overlap and, furthermore, the overlap must not exceed the number k of allowed splits. This is indeed what we can show and, moreover, such graphs can be recognized in polynomial time. This finishes the outline of our characterization and we obtain:

▶ **Theorem 1.1** (★). *Let $\mathcal{F}$ be a set of graphs containing graphs of at most three vertices each. Then, $\text{Free}_{\prec}(\mathcal{F})$-VS is NP-complete if $\mathcal{F} = \{P_3\}$ or $\mathcal{F} = \{K_3\}$ and is in P otherwise.*

Our polynomial-time results for split- and threshold graphs (★) use the observation that destroying some of their forbidden subgraphs by splitting, namely P_4, C_4, or C_5, necessarily creates another forbidden subgraph $\overline{C_4}$, reducing the problem to checking whether the input graph has the respective property. This seems to be a general principle worthy of further exploration.

3 A General Framework to Show NP-hardness

In this section, we introduce a reduction framework and employ it to show NP-hardness of Π-VS characterized by a type of well-connected forbidden subgraphs. For each fixed $\ell \in \mathbb{N}$, consider the 2ℓ-Subdivided Cubic Vertex Cover problem: Given a tuple (G^*, k), where G^* is a 2ℓ-subdivision of a cubic graph G and $k \in \mathbb{N}$, is there a vertex cover C of G^* with $|C| \leq k$? The NP-hardness of this problem for each $\ell \in \mathbb{N}$ follows from a result by Uehara [34] and "folklore" techniques. Nevertheless, we provide a formal proof in the the full version of this paper [15].

Informally, our reduction works as follows: For a given set of forbidden subgraphs $\mathcal{F}$, to show the NP-hardness of either $\text{Free}_{\prec}(\mathcal{F})$-VS or $\text{Free}_{\subseteq}(\mathcal{F})$-VS, we reduce from 2ℓ-Subdivided Cubic Vertex Cover to the chosen problem. Here, ℓ will depend on the choice of $\mathcal{F}$.

Consider an instance (G, k) of the selected vertex cover problem. To build an instance of the vertex-splitting problem in question, we select some $H \in \mathcal{F}$ and designate two *"endpoint"* vertices of H. Then we replace each of G's edges with a copy of H (we call this copy an *edge gadget*) and keep k the same.

It is straightforward to see that, if one can split the constructed graph at most k times while destroying all forbidden graphs $\mathcal{F}$, one can find a corresponding vertex cover of G of size at most k: Analogous to how a vertex cover needs to "hit" each edge of G, the splits performed in the constructed graph need to destroy all the inserted forbidden copies of H.

The converse direction, that is, to show how a vertex cover of G can be used to destroy all forbidden subgraphs in the construction, is substantially more involved. Essentially, for each vertex in the vertex cover, we split in a particular way the corresponding vertex in the construction where edge gadgets meet (which we call an *attachment point*). This way, we can easily destroy all "original" embeddings of H. The hard part is to ensure that apart from these embeddings of H, by performing the construction and then the splits, no new embeddings of forbidden subgraphs of $\mathcal{F}$ are introduced.

To make the above outline precise, we first introduce the concept of a *splitting configuration*. Intuitively, a splitting configuration consists of a graph H, a selection of two of its vertices a and b, which we call H's *a-end* and *b-end*, and an encoding of a specific strategy of how to split a and b in H.

▶ **Definition 3.1.** *Let H be a graph, $a, b \in V(H)$ distinct vertices, $A_1, A_2 \subseteq N_H(a)$, and $B_1, B_2 \subseteq N_H(b)$, such that $A_1 \cup A_2 = N_H(a)$, $B_1 \cup B_2 = N_H(b)$, and A_1, A_2, B_1, B_2 are non-empty. Then, $(H, a, A_1, A_2, b, B_1, B_2)$ is called a* splitting configuration*. If $A_1 \cap A_2 = B_1 \cap B_2 = \varnothing$, we speak of a* disjoint splitting configuration*. Furthermore, we say the splitting configuration is* based on *$\mathcal{F}$ if $H \in \mathcal{F}$.*

Going forward, we aim to show how, depending on $\mathcal{F}$, one can find a suitable ℓ and a splitting configuration based on $\mathcal{F}$ such that the described reduction is guaranteed to be correct. We now make precise how to perform the construction. To specify which way (a-end, b-end) or (b-end, a-end) to insert the edge gadgets, we arbitrarily orient the graph G from the Vertex Cover instance to obtain a directed graph $\vec{G}$ as the "skeleton" graph. Furthermore, we also need a splitting configuration C (that also encodes the gadget-graph H and its a/b-ends to use). To simplify the correctness proof later on, we explain a more general construction than used in the reduction. That is, in addition, the construction takes a subset S of $\vec{G}$'s vertices as input; this set S specifies that the corresponding attachment points should be split in the construction according to the splitting configuration. When computing the reduction, we simply set $S = \varnothing$. See Figure 1 for a concrete example.

Towards defining $\text{Constr}(\vec{G}, C, S)$. Below, whenever we encounter a graph G' that is a copy of a graph G, we use $v^{G'}$ to denote the vertex that corresponds to $v \in V(G)$ in G'. We also do likewise for sets of vertices. Let $\vec{G}$ be a directed, oriented graph without loops, C a splitting configuration with $C = (H, a, A_1, A_2, b, B_1, B_2)$, and $S \subseteq V(G)$. We aim to define the graph $\text{Constr}(\vec{G}, C, S)$ and the map $\chi_{\text{Constr}(\vec{G},C,S)}$ which we will use to refer to particular subsets of vertices in the construction in the correctness proofs later on. For this we first define how to obtain the edge gadget graphs (H_e for each $e \in E(\vec{G})$), a map α that specifies

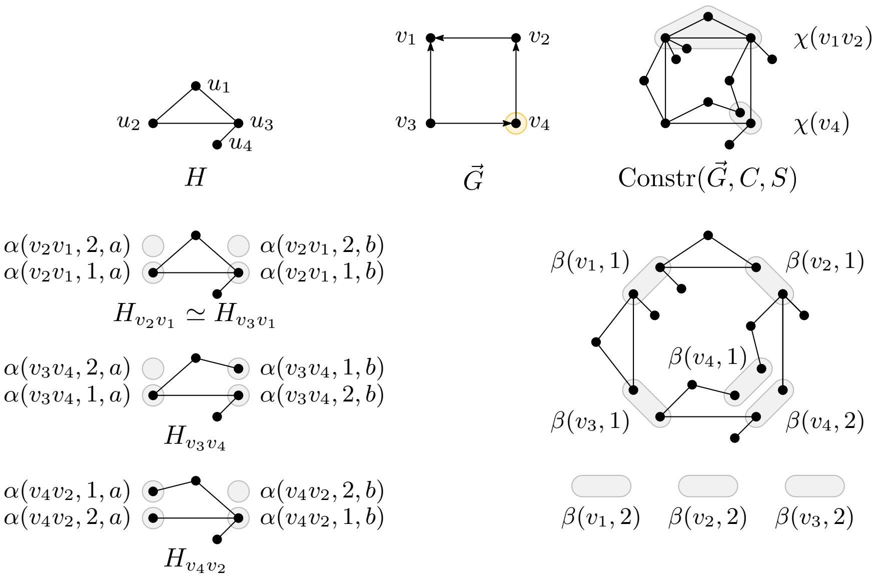


Figure 1 Example of Definition 3.2. The construction $\mathrm{Constr}(\vec{G}, C, S)$ is carried out for the "skeleton" graph $\vec{G}$, the splitting configuration C given by $(H, u_2, \{u_1\}, \{u_3\}, u_3, \{u_1\}, \{u_2, u_4\})$, and the set of vertices $S = \{v_4\}$ marked in yellow. The edge gadget graph is H; its "a-end" is u_2 and its "b-end" is u_3. The subscript of χ, $\mathrm{Constr}(\vec{G}, C, S)$, is dropped for brevity.

the attachment points inside the edge gadgets, and a map β that specifies which attachment points stemming from distinct edge gadgets should be merged to form the final attachment points where edge gadgets meet.

Towards defining H_e, with each arc $e = v_av_b \in E(\vec{G})$, we associate a fresh copy of H and call it H'_e. The vertices $a^{H'_e}, b^{H'_e}$ and the sets of vertices $A_1^{H'_e}, A_2^{H'_e}, B_1^{H'_e}, B_2^{H'_e}$ denote the corresponding vertex (resp. set of vertices) of H in its copy, H'_e. We obtain H_e by splitting a subset of $\{a^{H'_e}, b^{H'_e}\}$ in H'_e. Whether we split zero, one, or two vertices is dictated by S ($a^{H'_e}$ is split iff $v_a \in S$, $b^{H'_e}$ is split iff $v_b \in S$); the precise manner vertices are split is dictated by the splitting configuration C. More specifically, the neighborhoods of the descendant vertices of $a^{H'_e}$ (resp. $b^{H'_e}$) are given by $A_1^{H'_e}, A_2^{H'_e}$ (resp. $B_1^{H'_e}, B_2^{H'_e}$). With this, we can specify formally how H_e is obtained from each $e = v_av_b$ of $E(\vec{G})$:

$$H_e := \begin{cases} H'_e & \text{if } v_a \notin S, v_b \notin S,\\ \mathrm{Split}(H'_e, a^{H'_e}, A_1^{H'_e}, A_2^{H'_e}, a_1^{H_e}, a_2^{H_e}) & \text{if } v_a \in S, v_b \notin S,\\ \mathrm{Split}(H'_e, b^{H'_e}, B_1^{H'_e}, B_2^{H'_e}, b_1^{H_e}, b_2^{H_e}) & \text{if } v_a \notin S, v_b \in S, \text{ and}\\ \mathrm{Split}\big(\mathrm{Split}(H'_e, a^{H'_e}, A_1^{H'_e}, A_2^{H'_e}, a_1^{H_e}, a_2^{H_e}), & \\ \qquad b^{H'_e}, B_1^*, B_2^*, b_1^{H_e}, b_2^{H_e}\big) & \text{otherwise,} \end{cases}$$

where B_1^* (resp. B_2^*) denote the descendant vertices of $B_1^{H'_e}$ (resp. $B_2^{H'_e}$) with respect to the split described by $\mathrm{Split}(H'_e, a^{H'_e}, A_1^{H'_e}, A_2^{H'_e}, a_1^{H_e}, a_2^{H_e})$.[1] The set $\{H_e \mid e \in E(\vec{G})\}$ of edge gadgets provides the basic building blocks of $\mathrm{Constr}(\vec{G}, C, S)$. Note that the vertex sets of all H_e with $e \in E(\vec{G})$ are disjoint; to construct the final graph $\mathrm{Constr}(\vec{G}, C, S)$, we join the edge gadgets according to the structure of $\vec{G}$.

[1] This additional care is required to cover the case when a and b are neighbors in H.

For this purpose, we designate two numbered *attachment points* for the a-end, and two numbered attachment points for the b-end of each H_e, where an attachment point is a possibly empty subset of H_e's vertices. We denote the attachment points with the map $\alpha(\cdot,\cdot,\cdot)$, defined as follows: For a given $e \in E(\vec{G})$, $x \in \{a, b\}$, $i \in \{1, 2\}$ we set

$$\alpha(e, i, x) \coloneqq \begin{cases} \{x_i^{H_e}\} & \text{if } v_x \in S, \\ \{x^{H_e}\} & \text{if } v_x \notin S \wedge i = 1, \text{ and} \\ \varnothing & \text{if } v_x \notin S \wedge i = 2. \end{cases}$$

To join the edge gadgets, we define two equivalence classes for each $v \in V(\vec{G})$, stemming from the circumstance that we have two attachment points per edge gadget end. The set of equivalence classes is given by $\bigcup_{v \in V(\vec{G})}\{\beta(v, 1), \beta(v, 2)\}$, where for each $v \in V(\vec{G})$ and $i \in \{1, 2\}$, we define

$$\beta(v, i) \coloneqq \left\{ \left(\bigcup_{u \in N^-_{\vec{G}}(v)} \alpha(uv, i, b) \right) \cup \left(\bigcup_{u \in N^+_{\vec{G}}(v)} \alpha(vu, i, a) \right) \right\}.$$

See Figure 1 for a concrete example of $\alpha(\cdot,\cdot,\cdot)$ and $\beta(\cdot,\cdot)$.

▶ **Definition 3.2.** *The graph* $\mathrm{Constr}(\vec{G}, C, S)$ *is built by composing all* $(H_e)_{e \in E(\vec{G})}$ *into a single graph and merging all equivalent vertices into one representative vertex each.*

Later on, we will need to refer to specific vertex-subsets of the construction: For $G^* = \mathrm{Constr}(\vec{G}, C, S)$ and a given edge $e \in E(\vec{G})$, we write $\chi(e)_{G^*}$ for the set of vertices in G^* that stem from e's edge gadget (including the descendants of the gadgets "attachment"-vertices which are in general not unique to e); For a given vertex $v \in V(\vec{G})$, we write $\chi(v)_{G^*}$ to refer to either the set of the single "attachment"-vertex in G^* corresponding to v if $v \notin S$, and the two descendants of said vertex otherwise. See Figure 1 for a concrete example of $\chi(\cdot)$. A formal definition of χ is provided in the full version of this paper [15].

Abstracting from a single instantiation of our construction, we also introduce notation to capture the class of all possible constructions based on a given splitting configuration and an undirected graph together with all of its vertex covers.

▶ **Definition 3.3.** *Let G be a simple graph and C a splitting configuration. Then, we write* $\mathrm{AllConstr}(G, C)$ *to describe the set of all graphs* $\mathrm{Constr}(\vec{G}, C, S)$*, where $\vec{G}$ is an orientation of G and $S \subseteq V(G)$ is a vertex cover of G.*

3.1 Proving the Correctness of the Reduction

In this subsection, we define the property of *admissibility* for a splitting configuration C and show that, when using an admissible splitting configuration for the construction, the reduction outlined above is correct. The next subsection then deals with finding admissible splitting configurations for various classes of hereditary properties.

The backward direction of the correctness proof, that is, extracting a vertex cover from a splitting sequence that destroys all forbidden subgraphs, is straightforward and works independently of the choice of C and ℓ (★). However, the forward direction, where we use a vertex cover to find a splitting sequence that destroys all forbidden subgraphs in the construction, is more difficult. Here, the choice of C and ℓ will matter. We are given a vertex cover of the "skeleton graph" $\vec{G}$ and split all of the attachment points in the construction according to a corresponding splitting configuration. In the final graph of the splitting sequence, the whole construction needs to be free of embeddings of forbidden (induced) subgraphs. This can be rephrased as two separate properties that a splitting configuration must guarantee when applying our construction to any conceivable instance of 2ℓ-Subdivided Cubic Vertex Cover and splitting it according to a vertex cover:

- There are no embeddings of forbidden (induced) subgraphs reaching from one edge gadget to a neighboring edge gadget.
- There are no embeddings of forbidden (induced) subgraphs contained entirely within any individual edge gadget.

In Definition 3.4, we formalize both these requirements. Note that the requirement on $\mathcal{F}$ to be of bounded diameter will serve to guarantee that a suitable L can be found.

▶ **Definition 3.4.** *Let $\mathcal{F}$ be a family of graphs of bounded diameter with $H \in \mathcal{F}$ and $C = (H, a, A_1, A_2, b, B_1, B_2)$ a splitting configuration. Furthermore, let $L := 2 \cdot \max_{F \in \mathcal{F}} \operatorname{diam}(F)$. Then, C is called* separating *for $\mathcal{F}$ if for all graphs G that are an L-subdivision of some cubic graph, we have*

$$\forall G^* \in \operatorname{AllConstr}(G, C)\colon \forall F \in \mathcal{F}\colon \forall \pi \in \operatorname{Emb}_{\subseteq}(F, G^*)\colon \exists e \in E(G)\colon \operatorname{Range}(\pi) \subseteq \chi_{G^*}(e).$$

Furthermore, if $\operatorname{AllConstr}(K_2, C) \subseteq \operatorname{Free}_{\subseteq}(\mathcal{F})$, we say that C is intra-edge embedding-free *for $\mathcal{F}$. Finally, the splitting configuration C is called* admissible *for $\mathcal{F}$ if it is both separating for $\mathcal{F}$ as well as intra-edge embedding-free for $\mathcal{F}$.*

If such an admissible splitting configuration is known to exist, the converse direction of the correctness proof is straightforward (★). The following NP-hardness result follows directly by combining both directions:

▶ **Lemma 3.5 (★).** *Let $\mathcal{F}$ be a family of graphs of bounded diameter and let C be a splitting configuration admissible for $\mathcal{F}$. Then, $\operatorname{Free}_{\preccurlyeq}(\mathcal{F})$-VS and $\operatorname{Free}_{\subseteq}(\mathcal{F})$-VS are NP-hard.*

3.2 Biconnected Forbidden Subgraphs and Beyond

We just established a method for obtaining NP-hardness for vertex-splitting problems, provided an appropriate admissible splitting configuration exists. This subsection addresses how to find such splitting configurations for the case of biconnected, triconnected, and 4-connected forbidden (induced) subgraphs.

Towards this goal, we define a last piece of notation: the *width* of a splitting configuration, denoted by $\operatorname{wdt}(\cdot)$, represents the minimum distance between the two descendants of a split endpoint, a and b, respectively, after H has been split according to the splitting configuration.

First, we deal with biconnectedness. To that end we show that, given a splitting configuration of a certain width that is not separating (for some family of graphs $\mathcal{F}$ of bounded diameter and circumference), we can derive a new splitting configuration of increased width (Lemma 3.6). Since we cannot apply this process ad infinitum (when restricted to $\mathcal{F}$ of bounded circumference), we will arrive at a separating splitting configuration (★).

▶ **Lemma 3.6.** *Let $\mathcal{F}$ be a family of biconnected graphs of bounded diameter and let $C^0 = (H^0, a^0, A^0_1, A^0_2, b^0, B^0_1, B^0_2)$ be a disjoint splitting configuration of finite width with $H^0 \in \mathcal{F}$ that is not separating for $\mathcal{F}$. Then, there exists a disjoint splitting configuration $C^1 = (H^1, a^1, A^1_1, A^1_2, b^1, B^1_1, B^1_2)$ with $H^1 \in \mathcal{F}$ of finite width satisfying $\operatorname{wdt}(C^1) > \operatorname{wdt}(C^0)$.*

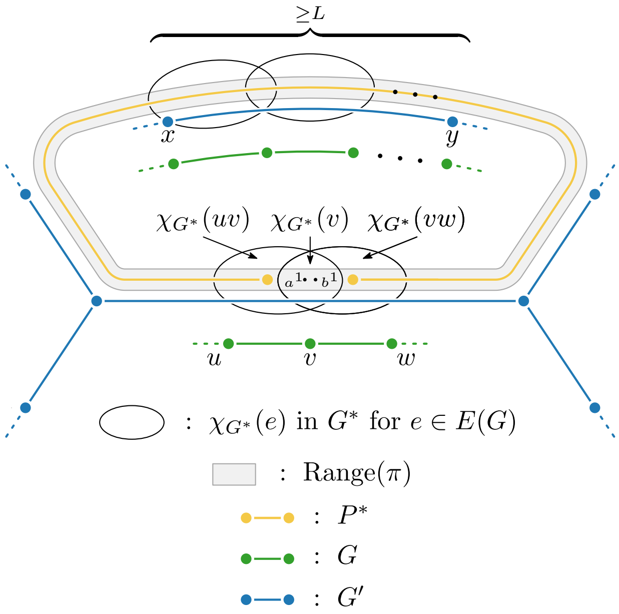


Figure 2 Illustration accompanying Lemma 3.6. The black ovals denote the edge gadgets in G^*, G is displayed in green, and the underlying graph G' is rendered in blue. The gray area shows the range of a hypothetical embedding π of F in G^*, that has to "go around" in the construction, since it cannot span across the intersection of edge gadgets $\chi_{G^*}(v)$. Additionally, in yellow, the path P^* traversing the embedding is shown.

Proof. As C^0 is not separating for $\mathcal{F}$, there is a graph G that is an L-subdivision of some cubic graph G', $G^* \in \text{AllConstr}(G, C^0)$, $F \in \mathcal{F}$ with $\pi \in \text{Emb}_{\subseteq}(F, G^*)$, as well as

$$L := 2 \cdot \max_{F \in \mathcal{F}} \text{diam}(F),$$

$$\text{Range}(\pi) \cap (\chi_{G^*}(uv) \setminus \chi_{G^*}(v)) \neq \varnothing, \text{ and}$$

$$\text{Range}(\pi) \cap (\chi_{G^*}(vw) \setminus \chi_{G^*}(v)) \neq \varnothing.$$

In other words, G^* is a graph constructed according to Definition 3.2 using a highly subdivided cubic graph (G) as a basis, where its edges were replaced by some forbidden graph $H \in \mathcal{F}$, and was split at the "attachment points" of edge gadgets according to some vertex cover of G' and the splitting configuration C^0. For this graph, we are provided a witness certifying that the splitting configuration C^0 is not separating with respect to $\mathcal{F}$ in the form of an embedding π of $F \in \mathcal{F}$ into G^*, where the embedding of F is not constrained to a single edge gadget, but rather uses vertices of at least two neighboring edge gadgets (of edges $uv, vw \in E(G')$), $\chi_{G^*}(uv)$ and $\chi_{G^*}(vw)$, such that the embedding is not entirely contained in the shared intersection $\chi_{G^*}(v)$. Notice that $\pi(\cdot)^{-1}$ refers to vertices of F, whereas $\pi(\cdot)$ refers to vertices of G^*. See Figure 2 for an illustration.

We now show that $\pi^{-1}(\chi_{G^*}(v) \cap \text{Range}(\pi))$ is a vertex separator of F, that is, if these vertices are deleted from F, the resulting graph is disconnected. We show this basically by observing that F can be embedded into G^* in a particular way (as witnessed by π), and since G^* has certain structural features, these carry over to F, leading to a contradiction.

Suppose that $\pi^{-1}(\chi_{G^*}(v) \cap \text{Range}(\pi))$ is not a vertex separator of F. Then, all neighbors of $\pi^{-1}(\chi_{G^*}(v) \cap \text{Range}(\pi))$ in $V(F) \setminus \pi^{-1}(\chi_{G^*}(v) \cap \text{Range}(\pi))$ are pairwise connected via some path in F not using any of $\pi^{-1}(\chi_{G^*}(v) \cap \text{Range}(\pi))$ each. Select any one of these paths and call it P. Without loss of generality, P starts with a vertex of $\pi^{-1}((\chi_{G^*}(uv) \setminus \chi_{G^*}(v)) \cap \mathrm{Range}(\pi))$ and ends in a vertex of $\pi^{-1}((\chi_{G^*}(vw) \setminus \chi_{G^*}(v)) \cap \mathrm{Range}(\pi))$. Due to the existence of π, we know that $P^* := \pi(P)$ gives an isomorphic path in G^*. Since P does not use vertices of $\pi^{-1}(\chi_{G^*}(v) \cap \mathrm{Range}(\pi))$, P^* does not use vertices of $\chi_{G^*}(v)$.

By construction of G^*, all paths connecting the first and last vertex of P^* in G^* that are constrained to the union of the vertex sets of both edge gadgets, that is to $\chi_{G^*}(uv) \cup \chi_{G^*}(vw)$, must traverse the intersection of both edge gadgets, that is, $\chi_{G^*}(uv) \cap \chi_{G^*}(vw) = \chi_{G^*}(v)$. But P^* does not intersect with $\chi_{G^*}(v)$, hence it is not one of these paths. Therefore, P^* must traverse G^* using edge gadgets the "other way around", that is, not use the direct connection.

Observe that P^* induces a path corresponding to the edge gadgets it traverses in G, which in turn induces a path of length at least three in the underlying cubic graph G'. At least one of these edges in G', say xy, must be fully traversed by P^* in the corresponding part of G^*. Thus, there are $x', y' \in V(P^*)$ where $x' \in \chi_{G^*}(x) \cap V(P^*)$ and $y' \in \chi_{G^*}(y) \cap V(P^*)$. The distance between x' and y' in G^* is at least $L = 2 \cdot \max_{F \in \mathcal{F}} \mathrm{diam}(F)$, the number of times xy is subdivided in G. But then $\pi^{-1}(x')$ and $\pi^{-1}(y')$, vertices of F, have distance of at least L in F as well, a contradiction to the choice of L. Thus, $\pi^{-1}(\chi_{G^*}(v) \cap \mathrm{Range}(\pi))$ is a vertex separator of F. Furthermore, since $|\chi_{G^*}(v)| \leq 2$ and F is biconnected, the vertex separator contains exactly two vertices. We shall denote its two elements by a^1 and b^1.

We continue exploiting the structure of F to obtain a splitting configuration satisfying the conditions of this lemma. Let D be any connected component of $F \setminus \{a^1, b^1\}$. Suppose there is only one edge of the form dv with $d \in V(D)$ and $v \in \{a^1, b^1\}$ in $E(F)$. Then, F could not be biconnected, for the removal of a single vertex (either a^1 or b^1) would suffice to render F disconnected. Thus, there is a path P^1 from a^1 to b^1 in F with $P^1 \subseteq V(D) \cup \{a^1, b^1\}$. Since G^* was constructed with respect to the splitting configuration C^0, we notice that $|P^1| \geq \mathrm{wdt}(C^0)$.

Let X be the vertex set of some distinct connected component of $F \setminus \{a^1, b^1\}$, and let $Y := V(F) \setminus (\{a^1, b^1\} \cup X)$. We notice that $a^1 b^1 \notin E(F)$, since $\pi(a^1)$ and $\pi(b^1)$ are descendants of the same split in the construction of G^*. Furthermore, X and Y form a partition of $V(F) \setminus \{a^1, b^1\}$. Thus, we may define a new disjoint splitting configuration C^1 as follows:

$$\begin{aligned}
&C^1 := (F, a^1, A^1_1, A^1_2, b^1, B^1_1, B^1_2), \text{ where}\\
&A^1_1 := N_F(a^1) \cap X,\\
&A^1_2 := N_F(a^1) \cap Y,\\
&B^1_1 := N_F(b^1) \cap X, \text{ and}\\
&B^1_2 := N_F(b^1) \cap Y.
\end{aligned}$$

Remember that by definition, the width of C^1 is $\min\{d_{F_1}(a^1_1, a^1_2), d_{F_2}(b^1_1, b^1_2)\}$, where $F_1 := \mathrm{Split}(F, a^1, A^1_1, A^1_2, a^1_1, a^1_2)$ and $F_2 := \mathrm{Split}(F, b^1, B^1_1, B^1_2, b^1_1, b^1_2)$, such that a^1_1, a^1_2, b^1_1, and b^1_2 are fresh vertices. Consider F_1: By the argument above, we deduce that there is a shortest path through the descendant vertices of X from a^1_1 to b^1 in F_1. Furthermore, since $F \setminus \{a^1, b^1\}$ is comprised of at least two connected components, there also exists a shortest path through one of them (using descendant vertices of Y) from b^1 to a^1_2. Each of the considered shortest paths must have length at least $\mathrm{wdt}(C^0)$, as G^* was constructed with respect to the splitting configuration C^0. Also, note that all paths connecting a^1_1 and a^1_2 in F must traverse b^1. Thus, combining these paths yields that $d_{F_1}(a^1_1, a^1_2) \geq 2\,\mathrm{wdt}(C^0)$. See Figure 3 for an illustration.

We proceed symmetrically for F_2. Hence, we obtain that $\mathrm{wdt}(C^1) > \mathrm{wdt}(C^0)$, and thus C^1 is a splitting configuration satisfying the required conditions. ◀

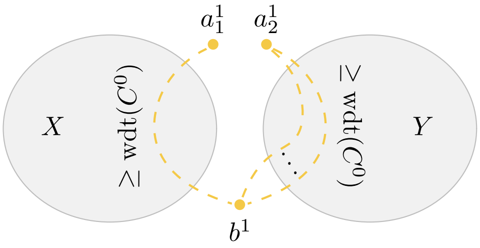


Figure 3 The derived splitting configuration C^1 has width at least $2\,\mathrm{wdt}(C^0)$.

It remains to ensure that the separating splitting configuration is additionally intra-edge embedding-free and therefore admissible. This property is implied when restricting $\mathcal{F}$ such that when any $F \in \mathcal{F}$ is destroyed by one or two non-trivial disjoint splits, the resulting graph is free of forbidden (induced) subgraphs. For example, each finite set of cycles satisfies this condition, or each set $\{F\}$ where F is biconnected (★). Finally, we can apply Lemma 3.5 to obtain NP-hardness of the corresponding vertex-splitting problems (★). In total, this then concludes the proof of the first part of Theorem 1.2, i.e., $\mathrm{Free}_{\prec/\subseteq}(\{F\})$-VS is NP-complete when F is biconnected. For the other parts, as we progress onward from biconnected graphs to higher degrees of connectedness, we can use similar techniques to show NP-hardness, but the restrictions imposed on the forbidden subgraphs relax. For the 4-connected case, no further restrictions are required.

4 Conclusion

In summary, for large families of graph classes Π, it is the case that Π-VS is NP-hard and, so far, nontrivial polynomial-time solvable cases are sporadic, such as Forest-VS and $\mathrm{Free}_{\prec}(\overline{K_3}, P_3)$-VS. Hence, the line of separation between tractability and intractability is much more jagged than in the case of Π Vertex Deletion, where a classical result by Lewis and Yannakakis shows the problem is NP-hard for hereditary Π if and only if Π is nontrivial, that is, Π and $\overline{\Pi}$ are infinite [27]. In contrast, the “complexity boundary” of Π-VS seems much more reminiscent of the classical Π Edge Deletion problem, for which no such characterization is known, despite extensive study since the late seventies.

Since for well-connected forbidden subgraphs our results imply hardness, a natural direction to further trace the line of separation between tractability and intractability would be to study more fragile forbidden subgraphs, that is, for instance, determining the complexity of $\mathrm{Free}_{\prec}(\{P_4\})$-VS and $\mathrm{Free}_{\prec}(\{K_{1,3}\})$-VS and seeing if patterns emerge in this regime. For the former, we can show a relation to a cograph-covering problem which we tend to believe is NP-hard. The latter we consider fully open. On the other hand, our results pave the way for studying broader notions of tractability instead of polynomial-time solvability such as approximating the optimal number of splits needed and further studying the parameterized complexity with respect to the number of splits.

In terms of approximation, our reduction for $\mathrm{Free}_{\prec}(\{K_3\})$-VS implies that minimizing the number of splits cannot be polynomial-time approximated to within an arbitrary fixed approximation factor, that is, there is no PTAS. However, constant-factor approximations may still exist and it would be interesting to see whether Π-VS is constant-factor approximable in polynomial-time if Π is characterized by a finite number of forbidden induced subgraphs or some large subfamily of such Π.

The parameterized complexity of Π-VS with respect to the number of splits also offers interesting contrasts and invites further investigation: $\mathrm{Free}_{\prec}(\{P_3\})$-VS is fixed-parameter tractable [14] but $\mathrm{Free}_{\prec}(\{K_3\})$-VS is para-NP-hard (Theorem 1.4). This raises the question to classify for which hereditary classes Π problem Π-VS is fixed-parameter tractable (analogous to the vertex-deletion version [26]). On the intractability side, it seems worthwhile to explore generalizations of the hardness construction for $\mathrm{Free}_{\prec}(\{K_3\})$-VS and constant number of splits per vertex (Theorem 1.4): The crucial property that we have exploited in the construction is that all constraints imposed by K_3s can only be solved by mapping edges between copies of the single vertex that we can feasibly split. If we use larger graphs instead of K_3, it is not obvious how to maintain this property. To obtain NP-hardness for a constant number of splits per vertex is it possible to replace K_3 by K_4, by K_ℓ for any fixed $\ell \geq 3$, or even a fixed graph of a more general graph class?

For tractability, it is tempting to exploit the connection to HITTING SET to try and obtain fixed-parameter tractability for Π characterized by a finite number of forbidden induced subgraphs. However, one has to work around two problems: First, the vertices to split are not necessarily a minimal hitting set (consider $\Pi = K_3$-free and the wheel graph of six vertices: the center vertex hits all forbidden triangles, yet at least two splits are needed to solve the instance). One thus has to efficiently find the additional split vertices that are not contained in an underlying minimal hitting set. Second, even after determining the vertices to split, one has to tackle interesting, often coloring-related problems such as in the case of $\mathrm{Free}_{\prec}(\{K_3\})$-VS.

Finally, it would be interesting to carry out a complexity classification program for Π-VS when Π is characterized by forbidden minors instead. An interesting starting point might be the contrast between the polynomial-time solvability of FOREST-VS, that is, vertex splitting to K_3-minor free graphs, and NP-hardness of PLANAR-VS, that is, K_5 and $K_{3,3}$-minor free graphs.

References

1 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In Jon Lee, Giovanni Rinaldi, and Ali Ridha Mahjoub, editors, *Proceedings of the 5th International Symposium of Combinatorial Optimization (ISCO 2018)*, volume 10856 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2018. doi:10.1007/978-3-319-96151-4_1.

2 Abu Reyan Ahmed, Stephen G. Kobourov, and Myroslav Kryven. An FPT algorithm for bipartite vertex splitting. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Proceedings of the 30th International Symposium Graph Drawing and Network Visualization (GD 2022)*, volume 13764 of *Lecture Notes in Computer Science*, pages 261–268. Springer, 2022. doi:10.1007/978-3-031-22203-0_19.

3 Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Blair D. Sullivan, and Petra Wolf. Cluster editing with overlapping communities. In Neeldhara Misra and Magnus Wahlström, editors, *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *LIPIcs*, pages 2:1–2:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.2.

4 Jakob Baumann, Matthias Pfretzschner, and Ignaz Rutter. Parameterized complexity of vertex splitting to pathwidth at most 1. In *Proceedings of the 49th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2023)*, volume 14093 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2023. doi:10.1007/978-3-031-43380-1_3.

5 Matthias Bentert, Alex Crane, Pål Grønås Drange, Felix Reidl, and Blair D. Sullivan. Correlation clustering with vertex splitting. In Hans L. Bodlaender, editor, *Proceedings of the 19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SWAT.2024.8.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *Journal of the ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

7 Vera Chekan and Stefan Kratsch. Tight algorithmic applications of clique-width generalizations. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *Proceedings of the 48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.35.

8 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

9 A. Davoodi, R. Javadi, and B. Omoomi. Edge clique covering sum of graphs. *Acta Mathematica Hungarica*, 149(1):82–91, 2016. doi:10.1007/s10474-016-0586-1.

10 Peter Eades and Candido Ferreira Xavier de Mendonça Neto. Vertex splitting and tension-free layout. In *Proceedings of the International Symposium on Graph Drawing (GD 1995)*, volume 1027 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 1995. doi:10.1007/BFb0021804.

11 David Eppstein, Philipp Kindermann, Stephen Kobourov, Giuseppe Liotta, Anna Lubiw, Aude Maignan, Debajyoti Mondal, Hamideh Vosoughpour, Sue Whitesides, and Stephen Wismath. On the planar split thickness of graphs. *Algorithmica*, 80:977–994, 2018. doi:10.1007/s00453-017-0328-y.

12 Luérbio Faria, Celina M. H. de Figueiredo, and Candido Ferreira Xavier de Mendonça Neto. SPLITTING NUMBER is NP-complete. *Discrete Applied Mathematics*, 108(1-2):65–83, 2001. doi:10.1016/S0166-218X(00)00220-1.

13 Alexander Firbas. Establishing hereditary graph properties via vertex splitting. Master's thesis, TU Wien, 2023. doi:10.34726/hss.2023.103864.

14 Alexander Firbas, Alexander Dobler, Fabian Holzer, Jakob Schafellner, Manuel Sorge, Anaïs Villedieu, and Monika Wißmann. The complexity of cluster vertex splitting and company. In Henning Fernau, Serge Gaspers, and Ralf Klasing, editors, *Proceedings of the 49th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2024)*, volume 14519 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2024. doi:10.1007/978-3-031-52113-3_16.

15 Alexander Firbas and Manuel Sorge. On the complexity of establishing hereditary graph properties via vertex splitting, 2024. arXiv:2401.16296, doi:10.48550/arXiv.2401.16296.

16 Herbert Fleischner. *Eulerian graphs and related topics*. North-Holland, 1990.

17 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In Alberto Pardo and Alfredo Viola, editors, *Proceedings of the 11th Latin American Symposium on Theoretical Informatics (LATIN 2014)*, volume 8392 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2014. doi:10.1007/978-3-642-54423-1_7.

18 Leslie Ann Goldberg and Marc Roth. Parameterised and fine-grained subgraph counting, modulo 2. *Algorithmica*, 86(4):944–1005, 2024. doi:10.1007/S00453-023-01178-0.

19 Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013. doi:10.1016/j.tcs.2012.12.041.

20 Jens Gramm, Jiong Guo, Falk Hüffner, Rolf Niedermeier, Hans-Peter Piepho, and Ramona Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(2):725–736, 2007. doi:10.1016/j.csda.2006.09.035.

21 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for bipartite contraction. *Information Processing Letters*, 113(22-24):906–912, 2013. doi:10.1016/j.ipl.2013.09.004.

22 Chengwei Guo and Leizhen Cai. Obtaining split graphs by edge contraction. *Theoretical Computer Science*, 607:60–67, 2015. doi:10.1016/j.tcs.2015.01.056.

23 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013. doi:10.1137/130907392.

24 Nathalie y Henr, Anastasia Bezerianos, and Jean-Daniel Fekete. Improving the readability of clustered social networks using node duplication. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1317–1324, 2008. doi:10.1109/TVCG.2008.141.

25 Anthony J. W. Hilton and C. Zhao. Vertex-splitting and chromatic index critical graphs. *Discrete Applied Mathematics*, 76(1-3):205–211, 1997. doi:10.1016/S0166-218X(96)00125-4.

26 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

27 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

28 Matthias Mayer and Fikret Erçal. Genetic algorithms for vertex splitting in DAGs. In *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA 1993)*, page 646. Morgan Kaufmann, 1993. URL: https://scholarsmine.mst.edu/comsci_techreports/25/.

29 George B. Mertzios and Derek G. Corneil. Vertex splitting and the recognition of trapezoid graphs. *Discrete Applied Mathematics*, 159(11):1131–1147, 2011. doi:10.1016/j.dam.2011.03.023.

30 Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD 2022)*, pages 232–246, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-22203-0_17.

31 Doowon Paik, Sudhakar M. Reddy, and Sartaj Sahni. Vertex splitting in dags and applications to partial scan designs and lossy circuits. *International Journal of Foundations of Computer Science*, 9(4):377–398, 1998. doi:10.1142/S0129054198000301.

32 Norbert Peyerimhoff, Marc Roth, Johannes Schmitt, Jakob Stix, Alina Vdovina, and Philip Wellnitz. Parameterized counting and Cayley graph expanders. *SIAM Journal on Discrete Mathematics*, 37(2):405–486, 2023. doi:10.1137/22M1479804.

33 W. T. Tutte. *Connectivity in graphs.* University of Toronto Press, 1966.

34 Ryuhei Uehara. NP-complete problems on a 3-connected cubic planar graph and their applications. *Tokyo Woman's Christian University, Tokyo, Japan, Tech. Rep. TWCU-M-0004*, 1996.

From Chinese Postman to Salesman and Beyond: Shortest Tour δ-Covering All Points on All Edges

Fabian Frei
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ahmed Ghazy
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarland University, Saarbrücken, Germany

Tim A. Hartmann
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Florian Hörsch
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

A well-studied continuous model of graphs, introduced by Dearing and Francis [Transportation Science, 1974], considers each edge as a continuous unit-length interval of points. For $\delta \geq 0$, we introduce the problem δ-Tour, where the objective is to find the shortest tour that comes within a distance of δ of every point on every edge. It can be observed that 0-Tour is essentially equivalent to the Chinese Postman Problem, which is solvable in polynomial time. In contrast, 1/2-Tour is essentially equivalent to the graphic Traveling Salesman Problem (TSP), which is NP-hard but admits a constant-factor approximation in polynomial time. We investigate δ-Tour for other values of δ, noting that the problem's behavior and the insights required to understand it differ significantly across various δ regimes. On the one hand, we first examine the approximability of the problem for every fixed $\delta > 0$:

(1) For every fixed $0 < \delta < 3/2$, the problem δ-Tour admits a constant-factor approximation and is APX-hard, while for every fixed $\delta \geq 3/2$, the problem admits an $O(\log n)$-approximation in polynomial time and has no polynomial-time $o(\log n)$-approximation, unless P = NP.

Our techniques also yield a new APX-hardness result for graphic TSP on cubic bipartite graphs. When parameterizing by the length of a shortest tour, it is relatively easy to show that 3/2 is the threshold of fixed-parameter tractability:

(2) For every fixed $0 < \delta < 3/2$, the problem δ-Tour is fixed-parameter tractable (FPT) when parameterized by the length of a shortest tour, while it is W[2]-hard for every fixed $\delta \geq 3/2$.

On the other hand, if δ is considered to be part of the input, then an interesting nontrivial phenomenon appears when δ is a constant fraction of the number of vertices:

(3) If δ is part of the input, then the problem can be solved in time $f(k)n^{O(k)}$, where $k = \lceil n/\delta \rceil$; however, assuming the Exponential-Time Hypothesis (ETH), there is no algorithm that solves the problem and runs in time $f(k)n^{o(k/\log k)}$.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Chinese Postman Problem, Traveling Salesman Problem, Continuous Graphs, Approximation Algorithms, Inapproximability, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.31

Related Version *Full Version*: https://arxiv.org/abs/2410.10613

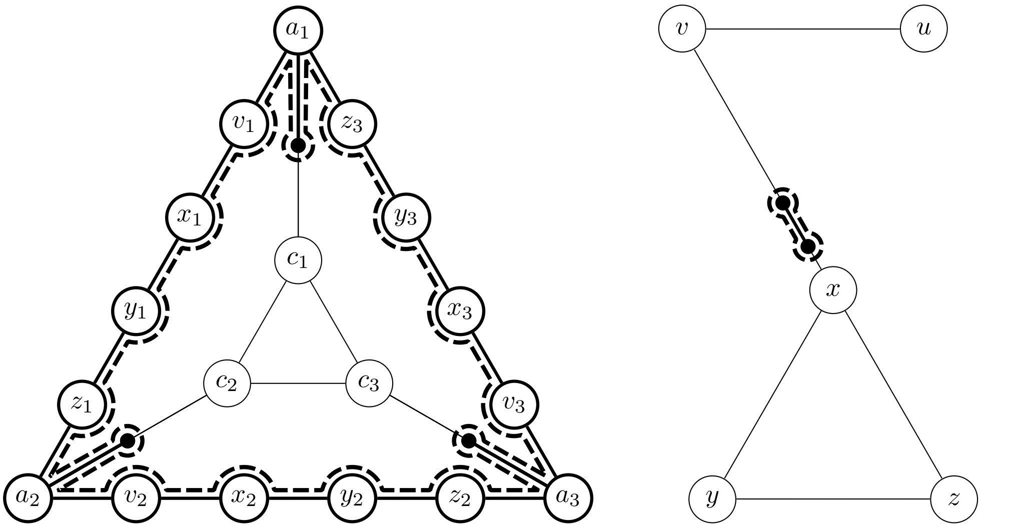
1 Introduction

We consider a well-studied continuous model of graphs introduced by Dearing and Francis [4]. Each edge is seen as a continuous unit interval of points with its vertices as endpoints. For any given graph G, this yields a compact metric space $(P(G), \mathrm{d})$ with a point set $P(G)$ and a distance function $\mathrm{d}\colon P(G)^2 \to \mathbb{R}_{\geq 0}$.

A prototypical problem in this setting is δ-Covering, introduced by Shier [28] for any positive real δ. The task is to find in G a minimum set S of points that δ-*covers* the entire graph, in the sense that each point in $P(G)$ has distance at most δ to some point in S. This problem, which is also often referred to as the continuous p-Center problem has been extensively studied; we cite only a few examples: [14, 2, 24]. Observe that the problem differs from typical discrete graph problems in two ways: the solution has to δ-cover every point of every edge (not just the vertices) and the solution may (and for optimality sometimes must) use points inside edges. How does the complexity of this problem depend on the distance δ? First, the problem is polynomial-time solvable when δ is a unit fraction, i.e., a rational with numerator 1, and NP-hard for all other rational and irrational δ [10, 13]. One can show that VertexCover is reducible to 2/3-Covering and DominatingSet is reducible to 3/2-Covering. Thus δ-Covering behaves very differently for different values of δ and can express problems of different nature and complexity: for example, while vertex cover is fixed-parameter tractable (FPT) when parameterized by the solution size, dominating set is W[2]-hard. This is reflected also in the complexity of δ-Covering: at the threshold of $\delta = 3/2$, the parameterized complexity of the problem, parameterized by the size of the solution, jumps from FPT to W[2]-hard [13]. Similarly, δ-Covering allows a constant factor approximation for $\delta < 3/2$ and becomes log-APX-hard for $\delta \geq 3/2$ [11]. The problem dual to δ-Covering is δ-Dispersion, as studied for example by Shier and Tamir [28, 29]. The task is to place a maximum number of points in the input graph such that they pairwise have distance at least δ from each other. For this problem, $\delta = 2$ marks the threshold where the parameterized complexity for the solution size as the parameter jumps from FPT to W[1]-hard [12]. Furthermore, the problem is polynomial-time solvable when δ is a rational with numerator 1 or 2, and NP-hard for all other rational and irrational δ [9, 12]. With δ-Covering being the continuous version of VertexCover and DominatingSet, and δ-Dispersion being a continuous version of IndependentSet, we now turn to the natural continuous variant of another famous problem.

We study the graphic Traveling Salesman Problem (TSP) with a positive real covering range δ in the continuous model, which we call δ-Tour. A δ-tour T is a tour that may make U-turns at arbitrary points of the graph, even inside edges, and is δ-covering, that is, every point in the graph is within distance δ from a point T passes by. The task in our problem δ-Tour is to find a shortest δ-tour. See Figure 1 for two examples of δ-tours that cannot be described as graph-theoretic closed walks. Note that computing a shortest 0-tour is equivalent to computing a shortest Chinese Postman tour (a closed walk going through every edge), which is known to be polynomial-time solvable [26, Chapter 29]. Moreover, one can observe that if every vertex of the input graph has degree at least two, then there is a shortest 1/2-tour that visits every vertex and, conversely, any tour visiting every vertex is a 1/2-tour. Thus, finding a shortest 1/2-tour is essentially equivalent to solving a TSP problem on a graph, with some additional careful handling of degree-1 vertices.

Our Results. It turns out that finding a shortest δ-tour is NP-hard for all $\delta > 0$; hence, we present approximation algorithms. As is standard, an α-*approximation algorithm* is one that runs in polynomial time and finds a solution of value within a factor α of the optimum.



(a) A graph and a δ-tour for $\delta = 1$. The tour δ-covers the inner part of this graph by peeking into three edges up to the midpoint. These three peek points are highlighted as the thick dots. The depicted tour (the thick dashed line) has length 18, which is shortest.

(b) The depicted shortest δ-tour for $\delta = 5/3$ of length $2 \cdot 1/6$ travels between the two points on edge vx at distances $1/3$ and $1/6$ from x.

Figure 1 Two examples of a δ-tour in a graph. On the right, see the special case of a tour fully contained in an edge.

As our main approximation result, for every fixed $\delta \in (0, 3/2)$, we give constant-factor approximation algorithms for finding a shortest δ-tour. We list our results in Table 1 and plot the approximation ratio against δ in Figure 3. As the complementing lower bound, we prove APX-hardness for every fixed $\delta \in (0, 3/2)$. Theorem 1.1 summarizes the general behavior; more details follow.

▶ **Theorem 1.1** (Constant-Factor Approximation). *For every fixed $\delta \in (0, 3/2)$, the problem δ-*Tour *admits a constant-factor approximation algorithm and is* APX*-hard.*

The problem behaves very differently in the various regimes of δ, even within the range of $(0, 3/2)$, and we exploit connections to different problems for different values of δ:

Case $\delta \in (0, 1/6]$. There is a close relation between our problem and the ChinesePostmanProblem in this range, which gives a good approximation ratio. When δ approaches 0, our approximation ratio approaches 1. See Theorem 3.3.

Case $\delta \in (1/6, 33/40)$. When $1/6 < \delta < 1/2$, the problem can be reduced to solving TSP on metric instances, for which we can use Christofides' 3/2-approximation algorithm [3] to obtain the same approximation ratio for our problem. See Theorem 3.4.
A simplification of this approach for $\delta = 1/2$ allows us to use the better 7/5-approximation for Graphic TSP due to Sebő and Vygen [27]. See Theorem 3.5.
Finally, for $1/2 < \delta < 33/40$, it turns out that a 1/2-tour is a good approximation of a δ-tour. See Theorem 3.6.

Case $\delta \in [33/40, 3/2)$. The problem here is closely related to a variation of the VertexCover problem, some results on which we exploit in our approximation algorithms [1, 21]. See Theorems thm:approx:ub:one:threehalves and 3.9.

The APX-hardness results are most challenging for small values of δ, where we first prove a lower bound for a family of cycle-covering type of problems, which we call $(\alpha, \beta, \gamma, \kappa)$-CycleSubpartition; see Section 3.2. Our reduction developed for δ-Tour further directly implies a new result for graphic TSP, namely APX-hardness on cubic bipartite graphs. To the best of our knowledge, even for graphic TSP restricted to cubic graphs without the added restriction to bipartitness, there is only one APX-hardness result that unfortunately happens to be flawed [19, Thm. 5.4]. In particular, the proposed tour reconfiguration argument appears to split the original TSP tour into two disjoint ones. The issue seems to affect results in a series of other papers [7, 8, 17, 18, 16, 15]. Fortunately, our separate approach closes the gap and yields an even stronger hardness result.

▶ **Theorem 1.2.** TSP *is* APX*-hard even on cubic bipartite graphs.*

Once δ reaches $3/2$, the problem δ-Tour suddenly changes character: it becomes similar to DominatingSet, where only a logarithmic-factor approximation is possible, unless P = NP; see Theorem 3.11 and Theorem 3.14.

▶ **Theorem 1.3** (Logarithmic Approximation). *For every fixed $\delta \geq 3/2$, the problem δ-Tour admits an $O(\log n)$-approximation algorithm and has no $o(\log n)$-approximation algorithm unless* P = NP*.*

The above approximation ratio in fact depends on δ, which the big-O notation hides. Thus, if δ is not fixed and is rather given as an input, this approximation guarantee can be arbitrarily bad. We show that a polylogarithmic-factor approximation is fortunately still possible in that setting.

▶ **Theorem 1.4** (Polylogarithmic Approximation). *There is a polynomial-time algorithm that, given $\delta > 0$ and a graph G of order n, computes a $64(\log n)^3$-approximation of a shortest δ-*tour *of G.*

Furthermore, we study the problem parameterized by the solution size, which is the length of the δ-tour. As mentioned above, when $\delta \geq 3/2$, then δ-Covering becomes similar to DominatingSet and is W[2]-hard. Therefore, it is not very surprising that $\delta = 3/2$ marks the threshold for the parameterized complexity of δ-tour as well; see Section 3.4.

▶ **Theorem 1.5** (Natural Parameterization). *Computing a shortest length δ-tour, parameterized by the length of the tour, is* FPT *for every fixed $0 < \delta < 3/2$, and* W[2]*-hard for every fixed $\delta \geq 3/2$.*

It is much more surprising what happens when δ is really large, comparable to the number of vertices. For this to make sense, we have to again consider the problem of computing a shortest δ-tour when δ is part of the input. In this regime, the problem becomes somewhat similar to covering the whole graph with $k = \lceil \frac{n}{\delta} \rceil$ balls of radius δ, suggesting the problem to be solvable in large part by guessing k centers in $n^{O(k)}$ time. Indeed, we give an algorithm for computing a shortest δ-tour in this runtime, and show the exponent to be essentially optimal.

▶ **Theorem 1.6** (XP Algorithm for Parameter n/δ). *There is an algorithm, which, given a connected n-vertex graph G, computes a shortest δ-*tour *of G in $f(k) \cdot n^{O(k)}$ time where $k = \lceil n/\delta \rceil$.*

▶ **Theorem 1.7** (Hardness for Parameter n/δ). *There are constants $\alpha > 0$ and k_0 such that, unless* ETH *fails, for every $k \geq k_0$, there is no algorithm that, given an n-vertex graph, computes a shortest δ-*tour *in $O(n^{\alpha k/\log k})$ time where $k = \lceil n/\delta \rceil$. Moreover, the problem is* W[1]*-hard parameterized by k.*

Section 2 begins with formal notions including a thorough definition of a δ-tour. Then Section 3 gives an extended overview of our results.

2 Formal Definitions

General Definitions. For a positive integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. All graphs in this article are undirected, unweighted and do not contain parallel edges or loops. Let G be a graph. For a subset of vertices $V' \subseteq V(G)$, we denote by $G[V']$ the subgraph induced by V'. The neighborhood of a vertex u is $N_G(u) := \{v \in V(G) \mid uv \in E(G)\}$. We write uv for an edge $\{u, v\} \in E(G)$. We denote by ln the natural logarithm and by log the binary logarithm.

Problem Related Definitions. For a graph G, we define a metric space whose point set $P(G)$ contains, somewhat informally speaking, all points on the continuum of each edge, which has unit length. We use the word *vertex* for the elements in $V(G)$, whereas we use the word *point* to denote elements in $P(G)$. Note however, that each vertex of G is also a point of G.

The set $P(G)$ is the set of points $p(u, v, \lambda)$ for every edge $uv \in E(G)$ and every $\lambda \in [0, 1]$ where $p(u, v, \lambda) = p(v, u, 1 - \lambda)$; $p(u, v, 0)$ coincides with u and $p(u, v, 1)$ coincides with v. The *distance* of points p, q on the same edge uv, say $p = p(u, v, \lambda_p)$ and $q = p(u, v, \lambda_q)$, is $\mathrm{d}(p, q) = |\lambda_q - \lambda_p|$. The *edge segment* $P(pq)$ of p and q then is the subset of points $\{p(u, v, \mu) \mid \min\{\lambda_p, \lambda_q\} \leq \mu \leq \max\{\lambda_p, \lambda_q\}\}$. A *$pq$-walk* T between points $p_0 := p$ and $p_z := q$ is a finite sequence of points $p_0p_1 \ldots p_z$ where every two consecutive points lie on the same edge, that is, formally, for every $i \in [z]$ there are an edge $u_iv_i \in E(G)$ and $\lambda_i, \mu_i \in [0, 1]$ such that $p_{i-1} = p(u_i, v_i, \lambda_i)$ and $p_i = p(u_i, v_i, \mu_i)$. When p and q are not specified, we may simply write *walk* instead of pq-walk. The *length* $\ell(T)$ of a walk T is $\sum_{i \in [z]} \mathrm{d}(p_{i-1}, p_i)$. A pq-walk T, whose length is minimum among all pq-walks, is called *shortest*. The points in the sequence defining a walk are called its *stopping points*. The point set of T is $P(T) = \bigcup_{i \in [z]} P(p_{i-1}, p_i)$. For some $p \in P(T)$, we say that T *passes* p. The *distance between two points* $p, q \in P(G)$, denoted as $\mathrm{d}(p, q)$, is the length of a shortest pq-walk, and ∞ if no such walk exists. Further, let $\mathrm{d}(p, Q) = \min\{\mathrm{d}(p, q) \mid q \in Q\}$ for $p \in P(G)$ and $Q \subseteq P(G)$.

A *tour* T is a p_0p_z-walk with $p_0 = p_z$. For a real $\delta > 0$, a *δ-tour* is a tour where $\mathrm{d}(p, P(T)) \leq \delta$ for every point $p \in P(G)$. We study the following minimization problem.

Optimization Problem δ-Tour, where $\delta \geq 0$.

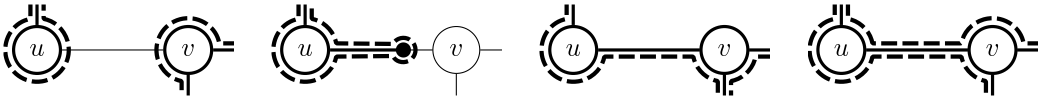
Instance A connected simple graph G.

Solution Any δ-tour T.

Goal Minimize the length $\ell(T)$.

Further, we use the following notions for a tour $T = p_0p_1 \ldots p_z$. A *tour segment* of T is a walk given by a contiguous subsequence of $p_0p_1 \ldots p_z$. The tour T *stops* at a point $p \in P(G)$ if $p \in \{p_0, p_1, \ldots, p_z\}$ and *traverses* an edge uv if uv or vu is a tour segment of T. The *discrete length* of a tour is z, that is, the length of the finite sequence of points representing it. We denote the discrete length of a tour T by $\alpha(T)$.

A point $p \in P(G)$ is *integral* if it coincides with a vertex. Similarly, $p = p(u, v, \lambda)$ is *half-integral* if $\lambda \in \{0, \frac{1}{2}, 1\}$.



(a) Staying out. **(b)** Peeking in. **(c)** Traversing once. **(d)** Traversing twice.

Figure 2 The four ways a nice δ-tour defined by at least 3 points can interact with an edge uv.

The *extension* of a tour $T = p_0p_1 \dots p_z$, denoted as $\lceil T \rceil$, is the integral tour where, for every edge $uv \in E(G)$ and every $\lambda < 1$, every tour segment $up(u, v, \lambda)u$ is replaced by uvu. Fruther, the *truncation* of a tour T, denoted as $\lfloor T \rfloor$, is the integral tour where for every edge $uv \in E(G), \lambda < 1$, every tour segment $up(u, v, \lambda)u$ in T is replaced by u. We note that $P(\lfloor T \rfloor) \subseteq P(T) \subseteq P(\lceil T \rceil)$.

3 Overview of Results

Section 3.1 provides key technical insights. We present our approximation algorithms in Section 3.2 and our hardness results in Section 3.3. Finally we turn to the parameterized complexity results in Section 3.4. All details are provided in the full version of this paper.

3.1 Structural Results

Because TSP in the continuous model of graphs is studied in this paper for the first time, we need to lay a substantial amount of groundwork. Due to the continuous nature of the problem, it is not clear a priori how to check if a solution is a valid δ-tour, or whether it is possible to compute a shortest δ-tour by a brute force search over a finite set of plausible tours. We clarify these issues in this section. While some of the arguments are intuitively easy to accept, the formal proofs are delicate with many corner cases to consider; the reader might want to skip these proofs at first reading.

Sometimes, a δ-tour has to make U-turns inside edges to be shortest; see Figure 1a. Indeed, it can be checked that an optimal 1-tour for the graph in Figure 1a must look exactly as depicted. Except for a single case, it is unnecessary for a tour to make more than one U-turn inside an edge. Indeed, the only case where a shortest tour is forced to make two U-turns in an edge is when the tour remains entirely within a single edge; see Figure 1b for an example. Note also that there exist degenerate cases in which a shortest δ-tour consists of a single point.

However, unless a tour is completely contained in a single edge, we can see that there are only four reasonable ways for a δ-tour to interact with the interior of any given edge, illustrated in Figure 2:

(a) completely avoiding the interior,
(b) peeking into the edge from one side,
(c) fully traversing the edge exactly once from one vertex to the other, or
(d) traversing the edge exactly twice.

We call a tour that restricts itself to this reasonable behavior a *nice* tour. The following result allows us to restrict our search space to nice tours.

▶ **Lemma 3.1** (Nice Shortest Tours). *Let G be a graph and δ a constant. Further, let a δ-tour T in G be given. Then, in polynomial time, we can compute a δ-tour T' in G with $\ell(T') \leq \ell(T)$ and such that all stopping points of T' are stopping points of T and T' is either nice or has at most two stopping points.*

Table 1 Approximation upper bounds (UB) and lower bound (LB) for δ-Tour.

$\boldsymbol{\delta}$	$(0, 1/6]$	$(1/6, 1/2)$	$1/2$	$(1/2, 33/40)$	$[33/40, 1)$	$[1, 3/2)$	$[3/2, \infty)$
UB	$1/(1-2\delta)$	1.5	1.4	$1.4/(2-2\delta)$	4	$3/(3-2\delta)$	$\min\{2\delta, 64\log^2 n\}\log n$
	Thm. 3.3	Thm. 3.4	Thm. 3.5	Thm. 3.6	Thm. 3.9	Thm. 3.8	Thms. 1.4 and 3.11
LB	APX-hard			APX-hard			$\Omega(\log n)$
	Thm. 3.12			Thm. 3.13			Thm. 3.14

Despite the continuous nature of δ-Tour, we show that we can actually study the problem in a discrete setting instead. More precisely, we prove that there is a nice shortest δ-tour T defined by points whose edge positions λ come from a small explicitly defined set. To show this, the idea is that there are only three scenarios for the edge position of a non-vertex stopping point p of T.

1. It has distance exactly δ to a vertex u. An example is that G is a long path with an end vertex u, and p is the closest point of $P(T)$ to u. Then p has an edge position λ that is the fractional part of δ.
2. It has distance exactly δ to a half-integral point $p(u, v, \frac{1}{2})$. An example is that G is a long ww'-path with a triangle uvw attached to w, and p is a closest point of $P(T)$ to $p(u, v, \frac{1}{2})$. Then p has an edge position λ which is the fractional part of $\delta + \frac{1}{2}$.
3. It has distance exactly 2δ to a vertex u. An example is that G contains a long uv-path P, p is the closest point of $P(T) - \{u\}$ on the path P to u, and T stops at u. Then p has an edge position λ which is the fractional part of 2δ.

Any δ-tour can efficiently be modified into one that is nice and has only such edge positions. Our technical proof uses some theory of linear programming, in particular some results on the vertex cover polytope.

▶ **Lemma 3.2** (Discretization Lemma). *For every $\delta > 0$ and every connected graph G, there is a shortest* δ-tour *such that each stopping point of the tour can be described as $p(u, v, \lambda)$ with $\lambda \in S_\delta$ where $S_\delta = \left\{0, \delta - \lfloor \delta \rfloor, \frac{1}{2} + \delta - \lfloor \frac{1}{2} + \delta \rfloor, 2\delta - \lfloor 2\delta \rfloor\right\}$.*

As a consequence, we can find a shortest δ-tour by a brute-force algorithm. Using some related arguments, we can check whether a given tour actually is a δ-tour in polynomial time.

3.2 Approximation Algorithms

Here, we overview the approximation algorithms we design for different ranges of δ. Most of our algorithms follow a general paradigm; our approach is to design a collection of *core* approximation algorithms for certain key values of δ and rely on one of the following two ideas to extrapolate the approximation ratios we get to previous and subsequent intervals. The first main idea uses the simple fact that a δ-tour is also $(\delta + x)$-tour where $x > 0$. Having an approximation algorithm for δ-Tour, if we are able to reasonably bound the ratio between the lengths of a shortest δ-tour and a shortest $(\delta + x)$-tour, we obtain an approximation algorithm for $(\delta + x)$-Tour essentially for free. The second main idea is complementary to the first. Namely, we show that we may also extend a given δ-tour to obtain a $(\delta - x)$-tour where $x > 0$. Again, having an approximation algorithm for the δ-Tour, if we have a good bound on the total length of the extensions we add, we obtain an approximation algorithm for $(\delta - x)$-Tour.

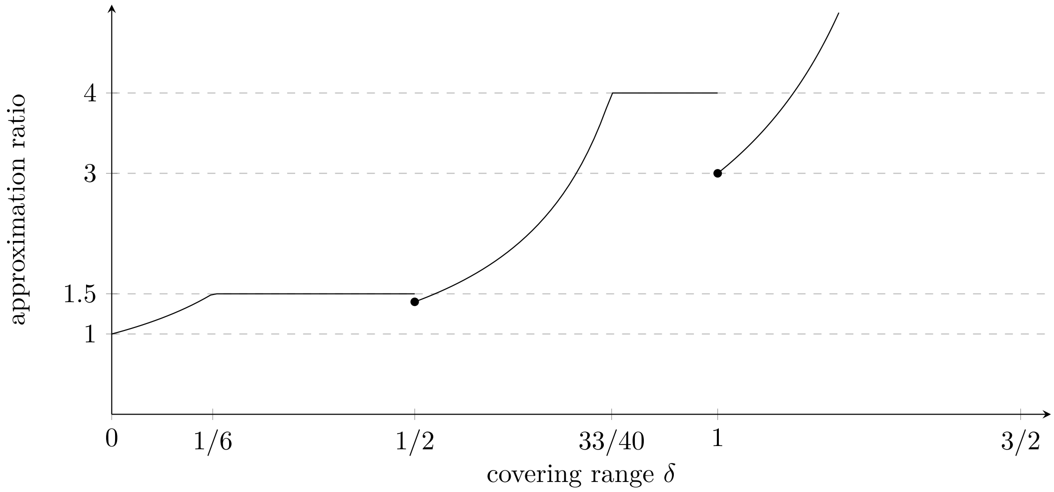


Figure 3 The approximation ratio of our algorithms for δ-Tour plotted against δ.

Approximation for $\delta \in (0, 1/6]$. The main idea is that a shortest Chinese Postman tour, that is, a tour which traverses every edge, is a good approximation of a δ-tour. Let us denote the length of a shortest δ-tour of a given graph by $\mathsf{OPT}_{\delta\text{-tour}}$ and the length of a shortest Chinese Postman tour by $\mathsf{OPT}_{\mathrm{CP}}$. To bound the ratio $\mathsf{OPT}_{\mathrm{CP}}/\mathsf{OPT}_{\delta\text{-tour}}$, we observe that there is a shortest δ-tour that, for every edge uv, either traverses uv or contains the segment of the form $up(u, v, \lambda)u$ for some $\lambda \in \{1 - \delta, 1 - 2\delta\}$. We obtain a Chinese Postman tour by replacing every such segment by a tour segment uvu. This bounds $\mathsf{OPT}_{\mathrm{CP}}/\mathsf{OPT}_{\delta\text{-tour}}$ by $1/(1 - 2\delta)$. Hence, outputting a shortest Chinese Postman tour, which can be computed in polynomial time [5], yields an approximation ratio of $1/(1 - 2\delta)$.

▶ **Theorem 3.3.** *For every $\delta \in (0, 1/6]$, δ-Tour admits a $1/(1-2\delta)$-approximation algorithm.*

Approximation for $\delta \in (1/6, 1/2)$. In this range, we rely on shortest δ-tours that satisfy certain desirable discrete properties. In the following more precise description, we focus on the case $\delta \leq \frac{1}{4}$, the construction needing to be slightly altered if $\frac{1}{4} < \delta \leq \frac{1}{2}$. Here, we prove the existence of a nice shortest δ-tour T such that

(P1) T contains the tour segment $up(u, v, 1 - \delta)u$ for every edge $uv \in E(G)$ incident to a leaf vertex v (that is, $\deg_G(v) = 1$) and

(P2) for every edge uv not incident to a leaf, either T traverses uv or the interaction of T with uv consists of one of the segments $up(u, v, 1 - 2\delta)u$ or $vp(v, u, 1 - 2\delta)v$.

We construct an auxiliary graph G' on the above listed points with edge weights corresponding to their distance in G. It turns out that TSP tours in G' are in a one-to-one correspondence with δ-tours in G satisfying properties (P1–P2). More precisely, we prove that an α-approximate TSP tour T' of G' can be efficiently turned into a δ-tour T of G of at most the same length which yields $\ell(T) \leq \ell(T') \leq \alpha \cdot \mathsf{OPT}_{\mathrm{TSP}}$, where $\mathsf{OPT}_{\mathrm{TSP}}$ denotes the length of a shortest TSP tour of G'. Then, noting that a given δ-tour of G satisfying properties (P1) and (P2) can be converted to a TSP tour of G' of at most the same length, we get that T is a δ-tour with $\ell(T) \leq \alpha \cdot \mathsf{OPT}_{\delta\text{-tour}}$. The first part, that is, proving that a TSP tour T' of G' can be turned into a δ-tour of G of the same length, is based on the fact that there is a limited number of ways a reasonable TSP tour interacts with the points corresponding to a certain edge. More precisely, any TSP tour in G' can easily be transformed into one which is not longer and whose interaction with the points in any edge is in direct correspondence with the interaction of a certain δ-tour with this edge in G.

This lets us transfer known positive approximation results for metric TSP to δ-tour. We may use the Christofides algorithm [3], yielding the following theorem.

▶ **Theorem 3.4.** *For every $\delta \in (1/6, 1/2)$, δ-Tour admits a 3/2-approximation algorithm.*

Approximation for $\delta = 1/2$. Even though the idea from the previous section still applies when $\delta = 1/2$, interestingly, we obtain a better approximation ratio observing that the problem further reduces to computing a graphic TSP tour on the non-leaf vertices, which admits a 1.4-approximation algorithm due to Sebő and Vygen [27].

▶ **Theorem 3.5.** 1/2-Tour *admits a 1.4-approximation algorithm.*

Approximation for $\delta \in (1/2, 33/40)$. In this range, we show that computing a 1/2-tour via Theorem 3.5 is a good approximation of a δ-tour. To that end, we characterize δ-tours for $\delta \in [1/2, 1]$, showing that, in particular, the existence of a shortest δ-tour T such that one of the following conditions hold for every edge uv.

(P1) T stops at both u and v, or

(P2) T stops at one of the endpoints, say u, and additionally stops at the point $p(u, v, \lambda)$ for some $\lambda \in [1 - \delta, 1]$, or T stops at the two points $p(u, v, \lambda_1)$ and $p(x, v, \lambda_2)$ for some $x \in N_G(v)$ where $\lambda_1 + \lambda_2 \geq 2 - 2\delta$, or

(P3) T stops at neither u nor v but stops at two points $p(x, v, \lambda_1)$ and $p(y, u, \lambda_2)$ for some $x \in N_G(v)$ and $y \in N_G(u)$ where $\lambda_1 + \lambda_2 \geq 3 - 2\delta$.

Let $\mathsf{OPT}_{1/2}$ and $\mathsf{OPT}_{\delta\text{-tour}}$ be the lengths of a shortest 1/2-tour and δ-tour in G, respectively. To bound the ratio $\mathsf{OPT}_{1/2}/\mathsf{OPT}_{\delta\text{-tour}}$, we observe that a given δ-tour T_δ can be transformed into a 1/2-tour $T_{1/2}$ by an appropriate replacement of every tour segment of T_δ corresponding to one of the cases (P1-3).

It can be shown that these modifications then result in a tour $T_{1/2}$ visiting every non-leaf vertex of G and covering leaves by tour segments of length 1, so $T_{1/2}$ is a 1/2-tour. These modifications increase the tour length by at most a multiplicative factor of $\max\{\frac{1}{2(1-\delta)}, \frac{2}{3-2\delta}\} = 1/(2 - 2\delta)$, so we have the following theorem.

▶ **Theorem 3.6.** *For every $\delta \in (1/2, 33/40)$, δ-Tour admits a $1.4/(2 - 2\delta)$-approximation algorithm.*

Approximation for $\delta \in (33/40, 3/2)$. Here we show that a 1-tour provides constant-factor approximations for δ-tours in this range. For $\delta > 1$, as in the previous range, due to a similar characterization of δ-tours, we can show an α-approximation algorithm for 1-Tour to imply an $\frac{\alpha}{3-2\delta}$-approximation algorithm. For $\delta < 1$, we show that starting from a 1-tour and augmenting it with some tour segments results in a δ-tour of a bounded length. The 3-approximation algorithm for a 1-Tour works as follows. We exploit a connection to the problem of computing a shortest *vertex cover tour*, which is a closed walk in a graph such that the vertices this tour stops at form a vertex cover. This problem, introduced in [1], admits a 3-approximation algorithm using linear programming (LP) techniques [21]. It is easy to see that a vertex cover tour forms a 1-tour; however, a shortest 1-tour can be shorter than a shortest vertex cover tour (e.g., this is the case in Figure 1a). Thus, the 1-tour we get from an arbitrary 3-approximation for vertex cover tour is in general not a 3-approximate 1-tour. Instead, we closely examine the LP formulated by Könemann et al. [21], showing the optimum for this LP to be a lower bound on the length of a 1-tour, which means that the vertex cover tours we get using this approach yield 3-approximate 1-tours.

Given a connected graph G, let $\mathcal{F}(G)$ be the set of subsets F of $V(G)$ such that both $G[F]$ and $G[V(G) - F]$ induce at least one edge. For a set $F \in \mathcal{F}(G)$, let $C_G(F)$ denote the set of edges in G with exactly one endpoint in F. The LP can then be formulated as follows:

$$\boxed{\begin{array}{lll} \text{Minimize} & \displaystyle\sum_{e \in E(G)} z_e & \\ \text{subject to} & \displaystyle\sum_{e \in C_G(F)} z_e \geq 2 & \text{for all } F \in \mathcal{F}(G) \text{ and} \\ 0 \leq z_e \leq 2 & \text{for all } e \in E(G). & \end{array}}$$

Denoting the optimal objective value of the above LP defined for a fixed graph G by $\mathsf{OPT}_{\mathrm{LP}}(G)$, the corollary below follows from [21].

▶ **Theorem 3.7** (Consequence of [21, Thms. 2 and 3]). *Given a connected graph G of order n, in polynomial time, we can compute a vertex cover tour T of G with $\ell(T) \leq 3 \cdot \mathsf{OPT}_{\mathrm{LP}}(G)$.*

It remains to show that $\mathsf{OPT}_{\mathrm{LP}}$ lower-bounds $\mathsf{OPT}_{\text{1-tour}}$, the length of a shortest 1-tour. Let $T_{\text{1-tour}} = p_0 \dots p_k$ $p_k = p_0$ be a nice 1-tour of G. For every edge $uv \in E(G)$, we define $\Lambda_{uv} := \sum_{i \in [k]: P(p_{i-1}p_i) \subseteq P(u,v)} d_G(p_{i-1}, p_i)$, indicating how much the tour $T_{\text{1-tour}}$ spends inside every edge uv. The vector $(\min(2, \Lambda_e))_{e \in E(G)}$ can then be shown to be feasible for the above LP. We observe the length of $T_{\text{1-tour}}$ to be at least $\sum_{e \in E(G)} \Lambda_e$, yielding $\mathsf{OPT}_{\text{1-tour}} \geq \mathsf{OPT}_{\mathrm{LP}}(G)$ and with Theorem 3.7, we obtain the following theorem.

▶ **Theorem 3.8.** *For any $\delta \in [1, 3/2)$, δ-Tour admits a $3/(3 - 2\delta)$-approximation algorithm.*

For the remaining range $\delta \in (33/40, 1)$, our algorithm first uses Theorem 3.8 to obtain a 3-approximate 1-tour T that is a vertex cover tour. Then, for every vertex $v \notin V(T)$, we choose an arbitrary neighbor u. Observe that $u \in V(T)$. Then we extend T into a tour T' by replacing an arbitrary occurence of u in T by the segment $up(u, v, 1 - \delta)u$ if v is a leaf vertex and by the segment $up(u, v, (1 - \delta))u$, otherwise. As T' fulfills the characterizing properties of a δ-tour, T' is a δ-tour. To bound its length, using our characterization, we observe that, given an arbitrary δ-tour T'', each non-leaf vertex v of G can be associated to a segment of T'' of cost at least $4(1 - \delta)$ as T'' either stops at v by traversing an edge, incurring a cost of at least 1, or makes two non-vertex stops that can be associated to v with a total cost of at least $2(2 - \delta)$. The previous observation can be used to show that the δ-tour we construct achieves an approximation ratio of 4.

▶ **Theorem 3.9.** *For any $\delta \in [33/40, 1)$, δ-Tour admits a 4-approximation algorithm.*

Approximation for $\delta > 3/2$. Here we design polylog(n)-approximation algorithms. We consider two different settings: one where δ is fixed and another where δ is part of the input. We show that each of the two problems can be reduced to an appropriate dominating set problem in an auxiliary graph. Recall that the discretization lemma (Lemma 3.2) shows, at a high-level, that there is a shortest δ-tour T of G whose stopping points on every edge come from a constant-sized set. Let $P_\delta(G)$ be the set of all such points in G.

In order to define our auxiliary graph, we first describe a collection $\mathcal{I}_{G,\delta}$ of edge segments of G. Namely, $\mathcal{I}_{G,\delta}$ is the collection of minimal edge segments each of whose whose endpoints is either a vertex of $V(G)$ or is of distance exactly δ to a point in $P_\delta(G)$ in G. This definition is suitable due to three properties of $\mathcal{I}_{G,\delta}$:

(P1) If T is a δ-tour in G whose stopping points are all contained in $P_\delta(G)$, then every $I \in \mathcal{I}_\delta(G)$ is fully covered by one stopping point of T,
(P2) every point in $P(G)$ is contained in some $I \in \mathcal{I}_{G,\delta}$, and
(P3) the number of segments in $\mathcal{I}_{G,\delta}$ is polynomial in n.

We are now ready to describe the auxiliary graph $\Gamma(G,\delta)$. We let $V(\Gamma(G,\delta))$ consist of $P_\delta(G)$ and a vertex x_I for every $I \in \mathcal{I}_{G,\delta}$. We further let $E(\Gamma(G,\delta))$ contain edges such that $\Gamma(G,\delta)[P_\delta(G)]$ is a clique and let it contain an edge px_I for $p \in P_\delta(G)$ and $I \in \mathcal{I}_{G,\delta}$ whenever p covers all the points in I. The main connection between δ-tours in G and dominating sets in $\Gamma(G,\delta)$ is due to the following lemma, which we algorithmically exploit in both settings, when $\delta \geq 3/2$ is fixed and when δ is part of the input.

▶ **Lemma 3.10.** *Let G be a graph and $\delta > 1$. Further, let T be a tour in G whose stopping points are all in $P_\delta(G)$. Then T is a δ-tour in G if and only if the stopping points of T are a dominating set in $\Gamma(G,\delta)$.*

Approximation for fixed $\delta > 3/2$. By computing a dominating set Y in the auxiliary graph $\Gamma(G,\delta)$ using a standard $\log n$-approximation algorithm and connecting it into a tour of length $O(\delta|Y|)$, we obtain the main result in this setting.

▶ **Theorem 3.11.** *For any $\delta \geq 3/2$, δ-Tour admits a $O(\log n)$-approximation algorithm.*

Approximation for δ as Part of the Input. The approach from the previous section does not yield any non-trivial approximation guarantee in this setting mainly because we get an additional factor of roughly δ when connecting the dominating set into a tour. However, we are able to show that a polylog(n)-approximation is attainable when δ is part of the input. The algorithm for this is based on a reduction to another problem related to dominating sets. Namely, a dominating tree U of a given graph H is a subgraph of H which is a tree and such that $V(U)$ is a dominating set of H. Kutiel [22] proves that given an edge-weighted graph H, we can compute a dominating tree of H of weight at most $\log^3 n$ times the minimum weight of a dominating tree of H.

In order to make use of this result, we now endow $E(\Gamma(G,\delta))$ with a weight function w. For all $p, p' \in P_\delta(G)$, we set $w(pp') = \mathrm{d}_G(p,p')$, and all other edges get a very large weight. We now compute an approximate dominating tree U of $\Gamma(G,\delta)$ with respect to w. By the definition of w, we obtain that U does not contain any vertex of $V(\Gamma(G,\delta)) - P_\delta(G)$. It follows that we can obtain a tour T from U that visits all points of $V(U)$ and whose weight is at most $2w(U)$. By Lemma 3.10, we obtain that T is a δ-tour in G.

Finally, in order to determine the quality of T, consider a shortest δ-tour T^* in G. It follows from Lemma 3.10 that the set P_{T^*} of points of $P_\delta(G)$ passed by T^* forms a dominating set of $\Gamma(G,\delta)$. Further, we can easily find a tree in $\Gamma(G,\delta)$ spanning P_{T^*} whose weight is at most the length of T^*. Hence, this tree is a dominating tree in $\Gamma(G,\delta)$, and Theorem 1.4 follows.

3.3 Inapproximability Results

Having presented our approximation algorithms providing a constant-factor approximation for every $\delta > 0$, we now rule out the existence of a PTAS by showing APX-hardness for every $\delta > 0$. The main challenge is to show the hardness for the range $\delta \in (0, 1/2]$. A simple subdivision argument then allows us to extend the hardness result to any $\delta > 0$. Further, we describe a stronger inapproximability result for $\delta \geq 3/2$.

APX-Hardness for Covering Range $\delta \in (0, 1/2]$. As the first step towards the APX-hardness of δ-Tour in the range $\delta \in (0, 1/2]$, we introduce a new family of optimization problems called $(\alpha, \beta, \gamma, \kappa)$-CycleSubpartition, that is also interesting on its own.

Optimization Problem $(\alpha, \beta, \gamma, \kappa)$-CycleSubpartition, where $\alpha, \beta, \gamma, \kappa \in \mathbb{R}$ and $\alpha, \beta, \gamma > 0$.

Instance	A simple graph G.
Solution	Any set $\mathcal{C}$ of pairwise vertex-disjoint cycles in G.
Goal	Minimize $\alpha\vert\mathcal{C}\vert + \beta\vert V(G) - \bigcup_{C\in\mathcal{C}} V(C)\vert + \gamma\vert V(G)\vert + \kappa$.

This bi-objective problem asks us, roughly speaking, for any given graph, to cover as many vertices as possible with a family of as few vertex disjoint cycles as possible. The precise balance between the two opposed optimization goals is tuned by the problem parameters. In particular, α specifies the cost for each cycle in the solution and β for each vertex left uncovered. Disallowing uncovered vertices (or making them prohibitively expensive), yields the classical APX-hard minimization problem CyclePartition [25, Thm 3.1, Prob. (iv)].

The two remaining parameters γ and κ may appear artificial since their only immediate effect is to make any solution for a given graph G more expensive by the same cost $\gamma|V(G)|+\kappa$. They will prove meaningful, however, for our main goal of this section. Namely, we first establish APX-hardness for $(\alpha, \beta, \gamma, \kappa)$-CycleSubpartition for cubic graphs in the entire parameter range of $\alpha, \beta, \gamma, \kappa \in \mathbb{R}, \alpha, \beta, \gamma > 0$ and then show that on cubic graphs, for every $\delta \in (0, 1/2]$, we have that δ-Tour coincides with this problem for an appropriate choice of the parameters α, β, γ, and κ. The proof uses a reduction from VertexCover on cubic graphs, which is known to be APX-hard [19, Thm. 5.4].

For some fixed $\alpha, \beta, \gamma, \kappa \in \mathbb{R}$ with $\alpha, \beta, \gamma > 0$, given an instance G of cubic VertexCover, we create a cubic graph H which we view as an instance of $(\alpha, \beta, \gamma, \kappa)$-CycleSubpartition. It is not difficult to obtain a packing of cycles in H with the appropriate properties from a vertex cover in G. The other direction, that is, obtaining a vertex cover in G from a cycle packing in H is significantly more delicate. A collection of careful reconfiguration arguments is needed to transform an arbitrary cycle cover in H into one that is of a certain particular shape and not more expensive. Having a cycle cover of this shape at hand, a corresponding vertex cover in G can easily be found. As mentioned above, we now easily obtain APX-hardness of δ-Tour for any $\delta \in (0, 1/2]$ and even the previously unknown APX-hardness for cubic bipartite graphic TSP.

▶ **Theorem 3.12.** *On cubic bipartite graphs, δ-Tour is APX-hard for $\delta \in (0, 1/2]$.*

TSP is APX-hard even on cubic bipartite graphs.

With Theorem 3.12 at hand, we further easily obtain a hardness result for larger values of δ. Namely, observe that for any nonnegative integer k, any constant δ and any connected graph G, there is a direct correspondence between the δ-tours in G and the $k\delta$-tours in the graph obtained from G by subdividing every edge $k-1$ times, yielding the following result.

▶ **Theorem 3.13.** *The problem δ-Tour is APX-hard for any real $\delta > 0$.*

Stronger Inapproximability for Covering Range $\delta \geq 3/2$. For $\delta \geq 3/2$, we give a lower bound of roughly $\ln n$ on the approximation ratio, asymptotically matching our upper bound. Like for all our inapproximability results, we prove hardness for the decision version of the problem.

▶ **Theorem 3.14.** *Unless* $\mathrm{P} = \mathrm{NP}$*, for every* $\delta \geq 3/2$*, there exists an absolute constant* α_δ *such that there is no* P*-time algorithm that, given a connected graph* G *and a constant* K*, returns "yes" if* G *admits a* δ-tour *of length at most* K *and "no" if* G *does not admit a* δ-tour *of length at most* $\alpha_\delta \log(|V(G)|)K$*.*

We start from the inapproximability of DominatingSet on split graphs, implicitly given by Dinur and Steurer [6, Corollary 1.5]. Given a split graph G satisfying some nontriviality condition, we can construct a graph G' such that the minimum size of a dominating set of G and the length of a shortest δ-tour in G' are closely related.

3.4 Parameterized Complexity

We examine the problem's parameterized complexity for two parameters: tour length and n/δ.

Parameterization by Tour Length. For $\delta \geq 3/2$, W[2]-hardness follows by a reduction from DominatingSet on split graphs, namely the same as used to show inapproximability for $\delta \geq 3/2$. We complement this result by giving an FPT-algorithm when $\delta < 3/2$. In fact, we give an algorithm which allows δ to be part of the input and that is fixed-parameter tractable for δ and maximum allowed tour length α combined.

▶ **Theorem 3.15.** *There is an algorithm that, given a graph* G *and reals* $\delta \in (0, 3/2)$ *and* $\alpha \geq 0$*, decides whether* G *has a* δ-tour *of length at most* α *in* $f(\alpha, \delta) \cdot n^{O(1)}$ *time, for some computable function* f*.*

Our algorithm is based on a kernelization: we either correctly conclude that G has no δ-tour of length at most α, or output an equivalent instance of size at most $f(\alpha, \delta)$ for a computable function f. The key insight is a bound on the vertex cover size of $f(\alpha, \delta)$ for a computable function, assuming there exists a δ-tour of length at most α. Hence we may compute an approximation C of a minimum vertex cover, and reject the instance if C is too large. We partition the vertices in $V(G) - C$ by their neighborhood in the vertex cover C. Now if a set S of the partition has size larger than $f(\alpha, \delta)$ for a computable function f, then deleting a vertex of S yields an equivalent instance.

XP Algorithm for Large Covering Range. We here give an overview of the proof of Theorem 1.6. The crucial idea is that, if T is a δ-tour, then, while the length of T can be linear in n, there exists a set of points stopped at by this tour that covers all the points in $P(G)$ and whose size is bounded by a function depending only on k where $k = \frac{n}{\delta}$. Intuitively speaking, the remainder of the tour is needed to connect the points in this set, but not to actually cover points in $P(G)$. Therefore, these segments connecting the points in the set can be chosen to be as short as possible. These observations can be subsumed in the following lemma, crucial to the proof of Theorem 1.6.

▶ **Lemma 3.16.** *Let* G *be a connected graph of order* n*,* δ *a positive real,* $k = \lceil \frac{n}{\delta} \rceil$ *and suppose that* $n \geq 12k$*. Further, let* T *be a shortest* δ-tour *in* G*. Then there exists a set* $Z \subseteq P(G)$ *of points stopped at by* T *with* $|Z| \leq 12k$ *such that for every point* $p \in P(G)$*, we have* $\mathrm{d}_G(p, Z) \leq \delta$*.*

With Lemma 3.16 at hand, Theorem 1.6 follows easily. Assuming that the minimum size requirement in Lemma 3.16 is met, due to the discretization lemma (Lemma 3.2), there exists a shortest tour only stopping at points from a set of size $O(n^2)$. We then enumerate all subsets of size at most $12k$ and for each of these sets, compute a shortest tour passing through its elements and check whether it is a δ-tour. It follows from Lemma 3.16 that the shortest tour found during this procedure is a shortest δ-tour. In the proof of Lemma 3.16, we define Z as the union of two sets Z_1 and Z_2. The set Z_1 is an inclusion-wise minimal set of points stopped at by T that covers all points in $P(G)$ whose distance to T is at least $\frac{n}{2k}$. For each $z \in Z_1$, by definition, there exists such a point p_z for which $\mathrm{d}_G(p_z, z') > \delta$ holds for all $z' \in Z - z$. Now for every $z \in Z_1$, we choose a shortest walk from p_z to z. It turns out that these walks do pairwise not share vertices of $V(G)$ and each of them contains $O(\frac{n}{k})$ vertices of $V(G)$. It follows that $|Z_1| = O(k)$. We now walk along $\lfloor T \rfloor$ and, roughly speaking, create Z_2 by adding every $\frac{n}{3k}$-th vertex stopped at by $\lfloor T \rfloor$. It turns out that Z_2 covers all points in $P(G)$ whose distance to T is at most $\frac{n}{2k}$. Further, as the length of $\lfloor T \rfloor$ is bounded by $2n$, we have $|Z_2| = O(k)$.

W[1]-Hardness for Large Covering Range. Given the XP-time algorithm running in time $f(k) \cdot n^{O(k)}$ designed for the regime $\delta = \Omega(n)$, it is natural to ask whether there is an FPT-time algorithm for the same parameter $k := \lceil \frac{n}{\delta} \rceil$. The answer is no. We show W[1]-hardness and the running time of our algorithm for Theorem 1.6 to be close to optimal under the Exponential Time Hypothesis (ETH). The hardness is based on the fact that BinaryCSP on cubic constraint graphs cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ under ETH; see [20, 23]. (The exact formulation we use is stronger and gives a lower bound for every fixed k.) An instance of BinaryCSP is a graph G with k edges, where the nodes represent variables taking values from a domain $\Sigma = [n]$, and every edge is associated with a constraint relation $C_{i,j} \subseteq \Sigma \times \Sigma$ over the two variables i and j. The instance is *satisfiable* if there is an assignment to the variables $\mathcal{A} : V(G) \to \Sigma$ such that $(\mathcal{A}(i), \mathcal{A}(j)) \in C_{i,j}$ for every constraint relation $C_{i,j}$.

In our reduction, we construct k gadgets corresponding to the constraints, with each gadget having some number of portals. Each gadget has multiple possible states corresponding to the satisfying assignments of its two constrained variables. If two constraints share a variable, then the corresponding gadgets are connected by paths between appropriate portals. These connections ensure that the selected states of the two gadgets agree on the value of the variable. It is now easy to find a tour in the auxiliary graph given a satisfying assignment for the formula. On the other hand, the construction is designed so that all tours in the auxiliary graph are in a certain shape, allowing to obtain an assignment from them.

References

1 Esther M. Arkin, Magnús M. Halldórsson, and Refael Hassin. Approximating the tree and tour covers of a graph. *Inf. Process. Lett.*, 47(6):275–282, 1993. doi:10.1016/0020-0190(93)90072-H.

2 Ramaswamy Chandrasekaran and Arie Tamir. An $o((n \log p)^2)$ algorithm for the continuous p-center problem on a tree. *SIAM J. Algebraic Discret. Methods*, 1(4):370–375, 1980. doi:10.1137/0601043.

3 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3(1):20, March 2022. doi:10.1007/s43069-021-00101-z.

4 Perino M. Dearing and Richard Lane Francis. A minimax location problem on a network. *Transportation Science*, 8(4):333–343, 1974.

5 Reinhard Diestel. *Graph Theory.* Springer, 5th edition, 2017.

6 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

7 Lars Engebretsen and Marek Karpinski. Approximation hardness of TSP with bounded metrics. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata,*

Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings, volume 2076 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2001. doi:10.1007/3-540-48224-5_17.

8 Lars Engebretsen and Marek Karpinski. TSP with bounded metrics. *Journal of Computer and System Sciences*, 72(4):509–546, 2006. doi:10.1016/j.jcss.2005.12.001.

9 Alexander Grigoriev, Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. Dispersing obnoxious facilities on a graph. *Algorithmica*, 83(6):1734–1749, 2021. doi:10.1007/S00453-021-00800-3.

10 Tim A. Hartmann. *Facility location on graphs.* Dissertation, RWTH Aachen University, Aachen, 2022. doi:10.18154/RWTH-2023-01837.

11 Tim A. Hartmann and Tom Janßen. Approximating δ-covering (*to appear*). In *Approximation and Online Algorithms - 22nd International Workshop, WAOA 2024, Egham, United Kingdom, September 5-6, 2024, Proceedings*, Lecture Notes in Computer Science. Springer, 2024.

12 Tim A. Hartmann and Stefan Lendl. Dispersing obnoxious facilities on graphs by rounding distances. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *Proceeding of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *LIPIcs*, pages 55:1–55:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.MFCS.2022.55.

13 Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. Continuous facility location on graphs. *Math. Program.*, 192(1):207–227, 2022. doi:10.1007/S10107-021-01646-X.

14 Oded Kariv and S. Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

15 Marek Karpinski. Towards better inapproximability bounds for TSP: A challenge of global dependencies. In Adrian Kosowski and Igor Walukiewicz, editors, *Proceedings of the 20th International Symposium on Fundamentals of Computation Theory (FCT 2015)*, volume 9210 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2015. doi:10.1007/978-3-319-22177-9_1.

16 Marek Karpinski and Richard Schmied. Improved inapproximability results for the shortest superstring and the bounded metric TSP. https://theory.cs.uni-bonn.de/ftp/reports/cs-reports/2013/85339-CS.pdf. Accessed: 2024-04-01.

17 Marek Karpinski and Richard Schmied. On approximation lower bounds for TSP with bounded metrics. *CoRR*, abs/1201.5821, 2012. arXiv:1201.5821.

18 Marek Karpinski and Richard Schmied. Improved inapproximability results for the shortest superstring and related problems. In Anthony Wirth, editor, *Nineteenth Computing: The Australasian Theory Symposium, CATS 2013, Adelaide, Australia, February 2013*, volume 141 of *CRPIT*, pages 27–36. Australian Computer Society, 2013. URL: http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV141Karpinski.html.

19 Marek Karpinski and Richard Schmied. Approximation hardness of graphic TSP on cubic graphs. *RAIRO – Operations Research*, 49(4):651–668, 2015. doi:10.1051/ro/2014062.

20 Karthik C. S., Dániel Marx, Marcin Pilipczuk, and Uéverton S. Souza. Conditional lower bounds for sparse parameterized 2-CSP: A streamlined proof. *CoRR*, abs/2311.05913, 2023. doi:10.48550/arXiv.2311.05913.

21 Jochen Könemann, Goran Konjevod, Ojas Parekh, and Amitabh Sinha. Improved approximations for tour and tree covers. *Algorithmica*, 38(3):441–449, 2004. doi:10.1007/s00453-003-1071-0.

22 Gilad Kutiel. Hardness results and approximation algorithms for the minimum dominating tree problem. *CoRR*, abs/1802.04498, 2018. arXiv:1802.04498.

23 Dániel Marx. Can you beat treewidth? In *FOCS*, pages 169–179. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.18.

24 Nimrod Megiddo and Arie Tamir. New results on the complexity of p-center problems. *SIAM J. Comput.*, 12(4):751–758, 1983. doi:10.1137/0212051.

25 Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976. doi:10.1145/321958.321975.

26 A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

27 András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, pages 1–34, 2014.

28 Douglas R. Shier. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11(3):243–252, 1977. URL: http://www.jstor.org/stable/25767877.

29 Arie Tamir. Obnoxious facility location on graphs. *SIAM J. Discret. Math.*, 4(4):550–567, 1991. doi:10.1137/0404048.

When Can Cluster Deletion with Bounded Weights Be Solved Efficiently?

Jaroslav Garvardt
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Christian Komusiewicz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Abstract

In the NP-hard Weighted Cluster Deletion problem, the input is an undirected graph $G = (V, E)$ and an edge-weight function $\omega : E \to \mathbb{N}$, and the task is to partition the vertex set V into cliques so that the total weight of edges in the cliques is maximized. Recently, it has been shown that Weighted Cluster Deletion is NP-hard on some graph classes where Cluster Deletion, the special case where every edge has unit weight, can be solved in polynomial time. We study the influence of the value t of the largest edge weight assigned by ω on the problem complexity for such graph classes. Our main results are that Weighted Cluster Deletion is fixed-parameter tractable with respect to t on graph classes whose graphs consist of well-separated clusters that are connected by a sparse periphery. Concrete examples for such classes are split graphs and graphs that are close to cluster graphs. We complement our results by strengthening previous hardness results for Weighted Cluster Deletion. For example, we show that Weighted Cluster Deletion is NP-hard on restricted subclasses of cographs even when every edge has weight 1 or 2.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph clustering, split graphs, cographs, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.32

Funding *Jaroslav Garvardt*: Supported by the Carl Zeiss Foundation, Germany, within the project “Interactive Inference”.

1 Introduction

Graph-based clustering is one of the core applications of graphs in computer science. This has led to a vast number of different algorithms and problem formulations for this task. One of the most fundamental problem formulations in this context is Cluster Deletion. In this problem, we are given an undirected graph G and ask for a partition of its vertex set into cliques that maximizes the total number of edges inside the cliques. Cluster Deletion is NP-hard [26] which has motivated the application of algorithmic approaches for hard problems to Cluster Deletion. In particular, the parameterized complexity of Cluster Deletion has been intensively studied [3, 12, 14, 16, 19, 22, 28].

Another closely related line of research is to study the complexity of Cluster Deletion on restricted classes of input graphs.[1] On the positive side, it was shown, for example, that Cluster Deletion can be solved in polynomial time on subcubic graphs [22], on cographs [13], on the more general class of P_4-sparse graphs [4], on interval graphs [23], and on several classes that generalize split graphs [23]. On the negative side, it was shown, for example, that Cluster Deletion remains NP-hard on planar graphs [15], on P_5-free

[1] For a definition of all considered graph classes, see Section 2.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 32; pp. 32:1–32:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

chordal graphs [5], and on C_4-free graphs with maximum degree 4 [22]. For graph classes defined by a single forbidden induced subgraph with at most four vertices a dichotomy into NP-hard and polynomial-time solvable cases is known [16].

All in all, the complexity of Cluster Deletion is fairly well-understood by now. In many applications, however, we are interested in the edge-weighted version of the problem where the aim is to maximize the total weight of the edges inside the clusters. This problem, called Weighted Cluster Deletion, turns out to be NP-hard for many graph classes where Cluster Deletion is polynomial-time solvable. For example, it has been shown that Weighted Cluster Deletion is NP-hard on cographs [5] and on split graphs [5]. Perhaps surprisingly, the problem is even NP-hard on split graphs where the independent set contains three vertices and every vertex of the independent set is adjacent to every vertex of the clique [19]. A closer inspection of the NP-hardness proofs shows, however, that the reductions construct instances where the maximum edge weight is unbounded. For example, in the above-mentioned reduction for the restricted class of split graphs with a periphery of constant size, the maximum edge weight is n^2 where n is the number of vertices in the constructed instance. In the vein of the deconstruction of hardness proofs [21], this observation begs the question of whether one can solve instances with bounded weights more efficiently. In particular, we would like to either identify graph classes on which parameterizing the maximum edge weight t in the input graph leads to FPT-algorithms or strengthen previous NP-hardness results to also hold in the case when the maximum edge weight is constant.

Our Results. Our results are summarized in Table 1. In a nutshell, we show that we obtain FPT-algorithms with respect to t on graph classes where large cliques are rather well-separated from each other. The first example of such an FPT-algorithm for t is obtained for the class of split graphs. Another class for which we can show fixed-parameter tractability is the class of almost cluster graphs. More precisely, we provide an FPT-algorithm for the combined parameter $\mathrm{cvd} + t$, where cvd is the vertex deletion distance of the input graph to cluster graphs. We then generalize these results to obtain algorithms for graphs with a bounded treewidth modulator to cliques (this generalizes split graphs) and for graphs which are almost split cluster graphs, that is, for the combined parameter $\mathrm{scvd} + t$, where scvd is the vertex deletion distance to graphs where every connected component is a split graph.

On the negative side, we show that even very restricted cases of Weighted Cluster Deletion remain NP-hard when all edges have weight 1 or 2. More precisely, we show this hardness for complete tripartite graphs and complete unipolar graphs, a subclass of the previously considered stable-like and laminar-like graphs which allow for polynomial-time algorithms in the unweighted case [23].

In addition, we show that it is presumably impossible to strengthen our FPT-algorithms for the parameter t to polynomial-size problem kernels. Finally, we show that Weighted Cluster Deletion is NP-hard on proper interval graphs, albeit with unbounded edge weights.

Due to lack of space, some proofs, marked by (*), are deferred to a full version of this work.

2 Preliminaries and Basic Observations

Notation. For $a, b \in \mathbb{N}$ with $a \leq b$ we write $[a, b]$ for the set $\{i \in \mathbb{N} \mid a \leq i \leq b\}$ and $[n] := [1, n]$. We consider undirected graphs $G = (V, E)$ with vertex set V and edge set E. The *neighborhood* of a vertex $v \in V$ is defined as $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ and for a set

Table 1 An overview over the (parameterized) complexity of WCD on the considered graph classes. Here, nd denotes the neighborhood diversity of the input graph, and cvd denotes its cluster vertex deletion number. For the first three classes, FPT refers to parameterization by maximum edge weight t plus either nd or cvd, for the other classes FPT refers to parameterization by t.

Graph class	Unweighted	Weighted	Bounded Max Weight
nd ≤ 2	P [19]	NP-h [19]	FPT Corollary 4.8
nd ≥ 3	FPT [19]	NP-h [19]	NP-h Theorem 3.6
bounded cvd	FPT [14]	NP-h (for cvd $= 2$) [19]	FPT Theorem 5.3
interval	P [23]	NP-h [5]	NP-h Proposition 3.4
proper interval	P [5]	NP-h Theorem 3.5	?
complete unipolar	P [23]	NP-h [19]	NP-h Proposition 3.4
co-graph	P [13]	NP-h [5]	NP-h Proposition 3.4
split	P [5]	NP-h [5]	FPT Theorem 4.6

of vertices $S \subseteq V$, we define $N_G(S) := (\bigcup_{v\in S} N_G(v)) \setminus S$ and $N_G^{\cap}(S) := (\bigcap_{v\in S} N_G(v))$. The *closed neighborhood* of a vertex $v \in V$ is defined as $N_G[v] := N_G(v) \cup \{v\}$ and for a set of vertices $S \subseteq V$, we define $N_G[S] := \bigcup_{v\in S} N_G[v]$. When the graph G is clear from the context we may omit the subscript. The *degree* of a vertex v is $|N(v)|$. For a vertex set $S \subseteq V$ we write $E(S) := \{\{u,v\} \in E \mid u,v \in S\}$ and denote with $G[S] := (S, E(S))$ the *subgraph of G induced by S*. Moreover, we define $G - S := G[V \setminus S]$.

Graph Classes and Graph Parameters. A vertex set $K \subseteq V$ that induces a complete graph is called a *clique*. A vertex set $I \subseteq V$ that induces a graph with no edges is called an *independent set*. A graph $G = (V, E)$ is a *cluster graph* if each connected component of G consists of a clique. A graph is a *cograph* if it does not contain the P_4, the path on four vertices, as an induced subgraph. A graph is an *interval graph* if it is the intersection graph of some set of intervals on the real line. A graph is a *proper interval graph* if additionally, none of the intervals is properly contained in another one of the intervals. A graph $G = (V, E)$ is *unipolar* if V can be partitioned into C and P such that C is a clique and $G[P]$ is a cluster graph [11]. We refer to C as the *core* and to P as the *periphery* of G. Furthermore, G is *complete unipolar* if each vertex of C is adjacent to each vertex of P. A unipolar graph $G = (V, E)$ is a *split graph* if P is an independent set. Moreover, G is a *dense split graph* if each vertex of C is adjacent to each vertex of P. A graph G is a *split cluster graph* if every connected component is a split graph [6]. We note the following relations between these graph classes. A complete unipolar graph is a cograph and an interval graph but not necessarily a proper interval graph. Also, every cluster graph is a split cluster graph.

A vertex set $S \subseteq V$ is a *vertex cover* for a graph G if the graph $G - S$ does not contain any edges. The *vertex cover number* $\mathrm{vc}(G)$ is the size of a smallest vertex cover for G. We say that a vertex set $M \subseteq V$ is a *cluster modulator for G* if $G - M$ is a cluster graph. The *cluster vertex deletion number* $\mathrm{cvd}(G)$ of G is defined as the size of the smallest cluster modulator for G. The relation $\sim$ with $u \sim v$ if $N(u) = N(v)$ or $N[u] = N[v]$ is an equivalence relation and the *neighborhood diversity* $\mathrm{nd}(G)$ is the number of equivalence classes of this relation. The *treewidth* of a graph G, denoted $\mathrm{tw}(G)$, is a parameter that, informally speaking, measures how close the graph is to a tree. For a formal definition of treewidth refer to [8]. We make use of the fact that $\mathrm{vc}(G) \geq \mathrm{tw}(G)$ for every graph G. If the graph G is clear from context, we omit it from the parameter notation.

Parameterized Complexity. An algorithm for a parameterized problem L is an FPT-algorithm, if there is a computable function f such that for every instance (I, k) the algorithm decides in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time whether (I, k) is a yes-instance of L. A *polynomial (many-one) kernel* for L is an algorithm that computes for each instance (I, k) in polynomial time an equivalent instance (I', k') with $(|I'| + k') \in k^{\mathcal{O}(1)}$. A *polynomial Turing kernel* for L is an algorithm that decides whether a given instance (I, k) is a yes-instance of L in time $(|I| + k)^{\mathcal{O}(1)}$, when given access to an oracle that decides membership in L for any instance (I', k') with $(|I'| + k') \in k^{\mathcal{O}(1)}$ in constant time.

A *polynomial parameter transformation* is a reduction from a parameterized problem A to a parameterized problem B that transforms each instance (I, k) of A in polynomial time into an equivalent instance (I', k') of B with $k' \in k^{\mathcal{O}(1)}$. Note that polynomial parameter transformations are transitive. Moreover, if there is a polynomial parameter transformation from A to B, where the unparameterized versions of A and B are in NP, then A admits a polynomial (many-one/Turing) kernel if B admits a polynomial (many-one/Turing) kernel. The Exponential Time Hypothesis (ETH) conjectures that 3-SAT cannot be solved in $2^{o(|F|)}$ time where F is the input formula. For further background on these definitions, we refer the reader to the standard monographs [8, 10].

Clusterings and Formal Problem Definition. A *clustering* $\mathcal{C}$ of a graph $G = (V, E)$ is a partition of V into subsets $C_1, \dots, C_r$ such that each C_i is a clique. For a clustering $\mathcal{C}$ of $G = (V, E)$ we denote with $E(\mathcal{C})$ the set of edges with both endpoints in the same cluster of $\mathcal{C}$. Let $\omega : E \to \mathbb{N}$ be an edge-weight function. For a set of edges $E' \subseteq E$, we define $\omega(E') := \sum_{e \in E'} \omega(e)$ and denote with $\omega(\mathcal{C}) := \omega(E(\mathcal{C}))$ the weight of a clustering $\mathcal{C}$.

Weighted Cluster Deletion (WCD)
Input: A graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{N}$ and a nonnegative integer k.
Question: Is there a clustering $\mathcal{C}$ for G such that $\omega(\mathcal{C}) \geq k$?

Some of our algorithms use brute force to find a clustering for a small subgraph of the input graph and then extend this clustering in an optimal way. This is formalized as follows. Let $G = (V, E)$ be a graph, let $S \subseteq V$ and let $\mathcal{C}$ and $\mathcal{C}_S$ be clusterings for G and $G[S]$, respectively. We say that $\mathcal{C}$ *extends* the clustering $\mathcal{C}_S$ if for each cluster C of $\mathcal{C}_S$, there is a cluster $C' \in \mathcal{C}$ such that $C' \cap S = C$. That is, restricting the clustering $\mathcal{C}$ to the vertices of S results in $\mathcal{C}_S$. Note that $\mathcal{C}$ may contain clusters that contain no vertex of S.

Basic Observations. A clique C in a graph $G = (V, E)$ is called a *critical clique* if all vertices of C have the same closed neighborhood. Furthermore, C is a *closed critical clique* if additionally $N[C]$ is a clique in G.

▶ **Observation 2.1.** *Let $G = (V, E)$ be a graph, let $\omega : E \to [1, t]$ be an edge-weight function, and let C be a closed critical clique in G. Then, each optimal clustering for G and ω extends $\{C\}$.*

This can be seen as follows. Let $\mathcal{C}$ be a clustering for G which contains two distinct clusters A and B that both contain vertices of C. Note that $A \cup B$ is a clique in G, since $A \cup B \subseteq N[C]$ and C is a closed critical clique. Hence, replacing the clusters A and B by $A \cup B$ yields a better clustering.

Note that if we try to find a clustering $\mathcal{C}$ that extends a clustering $\mathcal{C}_S$ of some subgraph $G[S]$, then no edges between vertices of distinct clusters of $\mathcal{C}_S$ are contained in $\mathcal{C}$, due to the definition of extending clusterings.

▶ **Observation 2.2.** *Let $G = (V, E)$ be a graph, let $S \subseteq V$, let $\mathcal{C}_S$ be a clustering of $G[S]$, and let G' be the graph obtained from G by removing all edges between vertices of distinct clusters of $\mathcal{C}_S$. Then G and G' share the same clusterings that extend $\mathcal{C}_S$.*

Hence, in the following, whenever – for a WCD instance and a clustering $\mathcal{C}_S$ for some subgraph $G[S]$ – we search for a clustering that extends $\mathcal{C}_S$, we may implicitly assume that $G[S]$ is a cluster graph. In the following, we show that we can further merge each cluster of $\mathcal{C}_S$ to a single vertex by increasing the largest assigned edge weight.

Let $G = (V, E)$ be a graph and let C be a clique in G. Then, *merging* C in G results in the graph $G' = (V', E')$, where C is replaced by a single vertex $v_C \in C$ with neighborhood $N_{G'}(v_C) := N_G^{\cap}(C)$. In other words, the vertex v_C keeps only the common neighborhood of C as neighbors.

▶ **Definition 2.3.** *Let $G = (V, E)$ be a graph, let $\omega : E \to [1, t]$ be an edge-weight function, and let C be a clique in G. The* weighted merge of C in G *yields the graph $G' = (V', E')$ obtained from merging C in G with edge-weight function ω' defined as follows: For each edge $e \in E'$ which is not incident with v_C, set $\omega'(e) := \omega(e)$, and for each edge $e = \{v_C, w\} \in E'$, set $\omega'(\{v_C, w\}) := \sum_{v \in C} \omega(\{v, w\})$.*

Note that the largest edge weight assigned by ω' is at most $|C| \cdot t$ and that each edge not incident with v_C is assigned a weight of at most t by ω'.

Similarly, for a vertex set $S \subseteq V$ where $G[S]$ is a cluster graph with collection of connected components $\mathcal{C}_S$, we define the *clustering-merge of $\mathcal{C}_S$ for G and ω* as the consecutive merges of all clusters of $\mathcal{C}_S$ in G and ω. Note that the largest edge weight in the resulting instance is at most $t \cdot \max_{C \in \mathcal{C}_S} |C|$, since G contains no edge with endpoints in distinct clusters of $\mathcal{C}_S$.

▶ **Lemma 2.4** (*). *Let $G = (V, E)$ be a graph, let $\omega : E \to [1, t]$ be an edge-weight function, and let $S \subseteq V$ such that $G[S]$ is a cluster graph with collection of connected components $\mathcal{C}_S$. Moreover, let $(G' = (V', E'), \omega')$ be the graph and edge-weight function obtained by the clustering-merge of $\mathcal{C}_S$ for G and ω. There is a bijection π between the clusterings of G' and the clusterings of G that extend $\mathcal{C}_S$, such that for every clustering $\mathcal{C}'$ of G'*

- *$\omega'(\mathcal{C}') = \omega(\pi(\mathcal{C}')) - \omega(\mathcal{C}_S)$ and*
- *$\pi(\mathcal{C}')$ extends $\mathcal{C}'$.*

The above lemma implies in particular, that we can obtain the optimal clustering of a graph G that extends some clustering $\mathcal{C}_S$ for some $G[S]$ by performing the clustering-merge and then computing the optimal clustering in the remaining instance.

3 Hardness Results

In this section, we present a reduction from Uniform Exact Cover to WCD on dense split graphs which implies several hardness results for WCD.

Uniform Exact Cover
Input: A set X and a collection $\mathcal{F}$ of size-d subsets of X.
Question: Is there a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that every element of X occurs in exactly one member of $\mathcal{F}'$?

Note that we can assume that $|X|$ is divisible by d, as otherwise the instance at hand is a trivial no-instance. Our reduction is a generalization of a known reduction from X3C [5], that is, Uniform Exact Cover with $d = 3$.

Construction. Let $I = (X, \mathcal{F})$ be an instance of Uniform Exact Cover, where d is the size of each set of $\mathcal{F}$ and $n := |X|$ is divisible by d. We construct from I an instance $I' = (G, \omega, k)$ of WCD as follows. Starting from an empty graph, we add to G a vertex v_x for each element $x \in X$ and make the set $V_X := \{v_x \mid x \in X\}$ a clique with $\omega(e) = 1$ for each edge e between two vertices in V_X. For each set $F \in \mathcal{F}$, we add a vertex v_F together with an edge $\{v_F, v_x\}$ with $\omega(\{v_F, v_x\}) = 2n$ for each $x \in F$ and an edge $\{v_F, v_x\}$ with $\omega(\{v_F, v_x\}) = n$ for each $x \in X \setminus F$. We define $V_{\mathcal{F}} := \{v_F \mid F \in \mathcal{F}\}$. Note that $V_{\mathcal{F}}$ is an independent set. We finish the construction by setting $k := n \cdot (2n + \frac{1}{d} \cdot \binom{d}{2})$.

Observe that the graph G in the construction above is a dense split graph with core V_X and periphery $V_{\mathcal{F}}$. Moreover, $t := 2n$ is the highest weight present in G and the core V_X consists of n vertices. Since the core of a split graph is a vertex cover, $t + \text{vc} \in \mathcal{O}(n)$, where vc denotes the vertex cover number of G. Moreover, G contains exactly $|X| + |\mathcal{F}|$ vertices.

Next, we show the equivalence of the two instances I and I'.

▶ **Lemma 3.1.** *I is a yes-instance of* Uniform Exact Cover *if and only if I' is a yes-instance of* WCD.

Proof. (⇒) Let I be a yes-instance and let $\mathcal{F}'$ be a solution for I. For each $F \in \mathcal{F}'$ let $C_F = \{v_x \in V_X \mid x \in F\} \cup \{v_F\}$. Consider the clustering $\mathcal{C} := \{C_F \mid F \in \mathcal{F}'\} \cup \{\{v_F\} \mid F \notin \mathcal{F}'\}$. Since $\mathcal{F}'$ is a solution for I, each $x \in X$ is covered by exactly one set $F \in \mathcal{F}'$ and thus each vertex of V_X is in exactly one set of $\mathcal{C}$. Note that V_X, and thus also each subset of V_X, is a clique and each vertex v_F is adjacent to each vertex in V_X. Therefore, each $C_F \in \mathcal{C}$ is a clique and $\mathcal{C}$ is a valid clustering. Moreover, we have $\omega(E(\mathcal{C})) = n \cdot (2n + \frac{1}{d} \cdot \binom{d}{2})$. Hence, I' is a yes-instance.

(⇐) Let I' be a yes-instance and let $\mathcal{C}$ be an optimal solution for I'. Let $E_{\mathcal{C}}(X)$ denote the edges that are preserved in $\mathcal{C}$ between vertices of V_X and let $E_{\mathcal{C}}(\mathcal{F})$ denote the edges that are preserved in $\mathcal{C}$ between a vertex of V_X and a vertex of $V_{\mathcal{F}}$. Clearly, $\omega(E(\mathcal{C})) = \omega(E_{\mathcal{C}}(X)) + \omega(E_{\mathcal{C}}(\mathcal{F}))$.

We now show that each $v_x \in V_X$ is part of a cluster containing a vertex $v_F \in V_{\mathcal{F}}$ such that $x \in F$. Let $x \in X$ and let $C_x \in \mathcal{C}$ be the cluster containing v_x. Note that C_x can contain at most one vertex of $V_{\mathcal{F}}$, since $V_{\mathcal{F}}$ is an independent set. This implies that there is at most one edge of $E_{\mathcal{C}}(\mathcal{F})$ incident with v_x. Suppose that C_x does not contain a vertex v_F such that $x \in F$. If an edge $\{v_x, v_{F'}\} \in E_{\mathcal{C}}(\mathcal{F})$ exists, it has weight at most n, since $x \notin F'$. Let $F \in \mathcal{F}$ be an arbitrary set such that $x \in F$ and let C_F denote the cluster of $\mathcal{C}$ that contains v_F. Moreover, let $\mathcal{C}'$ be the clustering obtained by moving vertex v_x in the cluster C_F. Note that $\mathcal{C}'$ is a valid clustering, since each vertex of V_X is adjacent to every other vertex of G, and that the weight of the edge $\{v_x, v_F\}$ is $2n$. Since $x \in F$ and v_x has at most $n-1$ neighbors in $C_x \cap V_X$, we have

$$\omega(\mathcal{C}') - \omega(\mathcal{C}) \geq \omega(\{v_x, v_F\}) - \sum_{w \in C_x \setminus \{v_x\}} \omega(\{v_x, w\}) \geq 2n - (n + |C_x \cap V_X|) > 0,$$

which is a contradiction to $\mathcal{C}$ being an optimal solution.

Thus, we can now assume that $\omega(E_{\mathcal{C}}(\mathcal{F})) = n \cdot 2n$. Furthermore, since each v_x is part of a cluster containing a vertex $v_F \in V_{\mathcal{F}}$ with $x \in F$, and each $F \in \mathcal{F}$ contains exactly d elements, at most d vertices from V_X can be in the same cluster in $\mathcal{C}$. More precisely, for each $x \in X$ let $C_x \in \mathcal{C}$ be the cluster containing v_x. We have $|(C_x \cap V_X) \setminus \{v_x\}| \leq d - 1$ for each $x \in X$.

Recall that edges between vertices of V_X have weight 1 and each edge in $E_{\mathcal{C}}(X)$ has two endpoints in V_X. If for some $x \in X$ we have $|(C_x \cap V_X) \setminus \{v_x\}| < d - 1$, then $\omega(E_{\mathcal{C}}(X)) = \frac{\sum_{x \in X} |(C_x \cap V_X) \setminus \{v_x\}|}{2} < \frac{n \cdot (d-1)}{2}$ and hence $\omega(E(\mathcal{C})) = \omega(E_{\mathcal{C}}(X)) + \omega(E_{\mathcal{C}}(\mathcal{F})) < \frac{n \cdot (d-1)}{2} + n \cdot 2n = k$, a contradiction to $\mathcal{C}$ being a solution.

Therefore, for each $x \in X$ we have $|(C_x \cap V_X) \setminus \{v_x\}| = d - 1$. Hence, each cluster C of $\mathcal{C}$ that contains at least one vertex of V_X fulfills $C = \{v_F\} \cup \{v_x \mid x \in F\}$ for some set $F \in \mathcal{F}$. Thus, the set $\mathcal{F}' := \{F \in \mathcal{F} \mid C_F \cap V_X \neq \emptyset, C_F \in \mathcal{C}, v_F \in C_F\}$ contains all elements in X exactly once and is a solution for I'. ◀

Lemma 3.1 and the bound on t and the vertex cover number of G directly give the following.

▶ **Proposition 3.2.** *There is a polynomial parameter transformation from* Uniform Exact Cover *with parameter n (size of the universe) to* WCD *on dense split graphs with parameter $t + \text{vc}$, where* vc *denotes the vertex cover number of the input graph.*

Due to Lemma 3.1 and the fact that Uniform Exact Cover cannot be solved in $2^{o(|X|+|\mathcal{F}|)} \cdot |I|^{\mathcal{O}(1)}$ time, unless the ETH fails [20][2], we also derive the following.

▶ **Corollary 3.3.** *Even on dense split graphs,* WCD *cannot be solved in $2^{o(n+t)} \cdot n^{\mathcal{O}(1)}$ time, unless the ETH fails.*

We can adapt the construction above in order to show that WCD is hard on a restricted subclass of cographs even if all edges have weight 1 or 2.

▶ **Proposition 3.4.** *On complete unipolar graphs* WCD *remains* NP*-hard even if $t = 2$ and cannot be solved in $2^{o(\text{cvd}+t)} \cdot n^{\mathcal{O}(1)}$ time, unless the ETH fails.*

Proof. To show the statement, we adapt the construction above so that we obtain from $I' = (G = (V, E), \omega, k)$ an equivalent instance $I'' = (G' = (V', E'), \omega', k')$ of WCD where G' is a complete unipolar graph and all edges have weight 1 or 2. We obtain I'' from I' as follows. Each vertex $v_F \in V_\mathcal{F}$ is replaced by a clique K_F of size n with $\omega'(\{u, v\}) = 1$ for each $u, v \in K_F$. We add edges such that K_F is fully connected to V_X and set $\omega'(\{u, v_x\}) = 2$ for each $u \in K_F$ and $v_x \in V_X$ with $\omega(\{v_F, v_x\}) = 2n$ as well as $\omega'(\{u, v_y\}) = 1$ for each $u \in K_F$ and $v_y \in V_X$ with $\omega(\{v_F, v_y\}) = n$. Moreover, we set $k' = k + |\mathcal{F}| \cdot \binom{n}{2}$. Note that in G' the sets K_F are disjoint cliques, each fully connected to the clique V_X. Therefore, G' is a complete unipolar graph. Furthermore, each edge in G' has weight either 1 or 2. Moreover, since G' is unipolar, V_X is a cluster modulator of size n. Hence, $\text{cvd}(G') \leq n$.

It remains to show that I' and I'' are equivalent instances. Observe that in G' for each $F \in \mathcal{F}$ the clique K_F is a closed critical clique and thus in every optimal clustering for G' each vertex of K_F is part of the same cluster due to Observation 2.1. Since the edges within these critical cliques all have weight 1, they have a total weight of $|\mathcal{F}| \cdot \binom{n}{2}$. Let $S := \bigcup_{F \in \mathcal{F}} K_F$ and let $\mathcal{C}_S := \bigcup_{F \in \mathcal{F}} \{K_F\}$. Per construction $\mathcal{C}_S$ is a clustering for $G'[S]$. Observe that (G, ω) can be obtained by the clustering-merge of $\mathcal{C}_S$ for G' and ω'. Clearly, $G'[S]$ is a cluster graph and each cluster K_F of $\mathcal{C}_S$ is merged into the vertex v_F in G with $N_G(v_F) = V_X = N_{G'}^{\cap}(K_F)$. Moreover, we have $\omega(\{v_F, v_x\}) = \sum_{v \in K_F} \omega'(\{v_F, v\})$.

By the above facts, Lemma 2.4 and Observation 2.1 imply that I' and I'' are equivalent instances. ◀

In addition, we obtain the following hardness results on graph classes that are unrelated to complete unipolar graphs.

▶ **Theorem 3.5 (*).** WCD *is* NP*-hard on proper interval graphs.*

[2] Note that Theorem 3.1 in [20] shows an ETH lower bound for 3D-Matching, which is a special case of Uniform Exact Cover.

▶ **Theorem 3.6** (*). WCD *is* NP*-hard on complete tripartite graphs even for* $t = 2$.

Note that complete tripartite graphs are cographs with neighborhood diversity 3. Hence, we obtain NP-hardness also for this class of graphs.

4 Split Graphs with Bounded Weights

We now show that WCD can be solved in FPT-time on split graphs when parameterized by the largest edge weight t. The algorithm is based on several properties concerning the interaction of the optimal clustering with the core of the split graphs. These properties will also be helpful for our algorithms on generalizations of split graphs.

As an auxiliary result, we first present an algorithm for WCD on split graphs, when parameterized by the size of the core of the split graph. The algorithm uses dynamic programming over subsets of the core.

▶ **Lemma 4.1.** *Let* $G = (V, E)$ *be a split graph with core* C *and let* ω *be an edge-weight function. One can find an optimal clustering for* G *and* ω *in* $3^{|C|} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. Let $P := \{p_1, \ldots, p_{|P|}\}$ be the periphery of G. We describe a dynamic program that finds an optimal clustering for G and ω in $3^{|C|} \cdot n^{\mathcal{O}(1)}$ time.

The dynamic programming table T has entries of type $T[i, S]$ for each $S \subseteq C$ and each $i \in [0, |P|]$ and stores the weight of an optimal clustering for $G[S \cup \{p_j \mid 1 \leq j \leq i\}]$. Hence, the base case for $i = 0$ and each $S \subseteq C$ is defined as $T[i, S] := \omega(E(S))$. This is correct, since C is a clique in G. For each other entry of the dynamic programming table, that is, for each $i \in [1, |P|]$ and each $S \subseteq C$, the recurrence to compute the entry $T[i, S]$ is

$$T[i, S] := \max_{\substack{S' \subseteq S \\ S' \subseteq N(p_i)}} \omega(E(S' \cup \{p_i\})) + T[i - 1, S \setminus S'].$$

Intuitively, we search for the best way to assign a subset S' of S to the cluster with vertex p_i and combine this with an optimal clustering for $G[(S \setminus S') \cup \{p_j \mid 1 \leq j < i\}]$ and ω. This is correct, since P is an independent set in G, and thus, in each valid clustering for G, the cluster containing p_i contains no other vertex of P. The total weight of an optimal clustering for G and ω is stored in $T[|P|, C]$ and a corresponding clustering can be found via traceback.

It thus remains to show the running time. Since each entry $T[i, S]$ can be computed in $2^{|S|} \cdot n^{\mathcal{O}(1)}$ time and there are $\binom{|C|}{\ell}$ subsets S of C of size exactly ℓ, all entries of the dynamic programming table T can be computed in $\sum_{\ell=0}^{|C|} \binom{|C|}{\ell} \cdot 2^\ell \cdot n^{\mathcal{O}(1)} = 3^{|C|} \cdot n^{\mathcal{O}(1)}$ time. ◀

Next, we show the first property for optimal clusterings of split graphs. Roughly speaking, the lemma shows that for each optimal clustering $\mathcal{C}$ and each subset $\mathcal{C}' \subseteq \mathcal{C}$, at least one of $\mathcal{C}'$ and $\mathcal{C} \setminus \mathcal{C}'$ contains at most $2t$ vertices of the core. This for example implies that the vertices of the core are only contained in $\mathcal{O}(t)$ clusters in any optimal clustering.

▶ **Lemma 4.2.** *Let* $G = (V, E)$ *be a split graph and let* $\omega : E \to [1, t]$ *be an edge-weight function. Moreover, let* $\mathcal{C}$ *be an optimal clustering for* G *and* ω*. Then, for each subset* $\mathcal{C}' \subseteq \mathcal{C}$*,* $\mathcal{C}'$ *contains at most* $2t$ *vertices of the core of* G *or* $\mathcal{C} \setminus \mathcal{C}'$ *contains at most* $2t$ *vertices of the core of* G*.*

Proof. Let C and P denote the core and the periphery of G, respectively. We show the contrapositive, that is, we show that $\mathcal{C}$ is not an optimal clustering for G and ω, if there is a subset $\mathcal{C}' \subseteq \mathcal{C}$, such that $\mathcal{C}'$ and $\mathcal{C} \setminus \mathcal{C}'$ contain at least $2t + 1$ vertices of the core C

each. To this end, we show that keeping the whole core as a cluster yields a better clustering than $\mathcal{C}$. That is, we show that $\mathcal{C}^* := \{C\} \cup \{\{p\} \mid p \in P\}$ is a better clustering than $\mathcal{C}$. Note that $\mathcal{C}^*$ is a valid clustering for G. It thus remains to show that $\omega(\mathcal{C}^*) - \omega(\mathcal{C}) > 0$. To this end, we first analyze the edges of $E(\mathcal{C}) \setminus E(\mathcal{C}^*)$. Since G is a split graph, each cluster of $\mathcal{C}$ contains at most one vertex of the periphery P. Hence, $E(\mathcal{C}) \setminus E(\mathcal{C}^*)$ contains for each vertex $v \in C$ at most one edge incident with v. Since each edge has weight at most t, this implies that $\omega(E(\mathcal{C}) \setminus E(\mathcal{C}^*)) \leq |C| \cdot t$. Next, we analyze the edges of $E(\mathcal{C}^*) \setminus E(\mathcal{C})$. Let C_1 denote the vertices of C that are contained in $\mathcal{C}'$ and let C_2 denote the vertices of C that are contained in $\mathcal{C} \setminus \mathcal{C}'$ and assume without loss of generality that $|C_1| \geq |C_2|$. Hence, $|C_1| \geq \frac{|C|}{2}$. Moreover, note that all edges of $E(C_1, C_2)$ are contained in $E(\mathcal{C}^*) \setminus E(\mathcal{C})$ which implies that $E(\mathcal{C}^*) \setminus E(\mathcal{C})$ contains at least $|E(C_1, C_2)| = |C_1| \cdot |C_2| \geq \frac{|C|}{2} \cdot (2t + 1) > |C| \cdot t$ edges of weight at least one each. Consequently,

$$\omega(\mathcal{C}^*) - \omega(\mathcal{C}) = \omega(E(\mathcal{C}^*) \setminus E(\mathcal{C})) - \omega(E(\mathcal{C}) \setminus E(\mathcal{C}^*)) > |C| \cdot t - |C| \cdot t = 0.$$

This implies that $\mathcal{C}$ is not an optimal clustering for G and ω. ◀

This has the following implications for any optimal clustering $\mathcal{C}$: There is at most one cluster of size more than $2t$ in $\mathcal{C}$ and there are $\mathcal{O}(t)$ clusters of size more than one in $\mathcal{C}$. We also derive the following.

▶ **Lemma 4.3.** *Let $G = (V, E)$ be a split graph with core C and let $\omega : E \to [1, t]$ be an edge-weight function. Moreover, let $\mathcal{C}$ be an optimal clustering for G and ω. If $|C| > 6t$, then there is a cluster $C^* \in \mathcal{C}$ that misses at most $2t$ vertices of C, that is, $|C^* \cap C| \geq |C| - 2t$.*

Proof. Let C^* be a cluster of $\mathcal{C}$ that contains the most vertices of C. First, we show that C^* contains at least $2t + 1$ vertices of C. Assume towards a contradiction that C^* contains at most $2t$ vertices of C, which then implies that every cluster of $\mathcal{C}$ contains at most $2t$ vertices of C. Let $\mathcal{C}'$ be an arbitrary subset of $\mathcal{C}$ such that $\mathcal{C}'$ contains at least $2t + 1$ vertices of C and no proper subset of $\mathcal{C}'$ contains at least $2t + 1$ vertices of C. Note that this implies that $\mathcal{C}'$ contains at most $4t$ vertices of C, since each cluster of $\mathcal{C}$ contains at most $2t$ vertices of C. Since C has size at least $6t + 1$, this implies that $\mathcal{C} \setminus \mathcal{C}'$ contains at least $2t + 1$ vertices of C. Due to Lemma 4.2, this contradicts the fact that $\mathcal{C}$ is an optimal clustering for G and ω. Hence, C^* contains at least $2t + 1$ vertices of C.

Now, consider the subset of clusters $\mathcal{C}' := \{C^*\}$. Since $\mathcal{C}$ is an optimal clustering for G and ω and $\mathcal{C}'$ contains more than $2t$ vertices of C, Lemma 4.2 implies that $\mathcal{C} \setminus \mathcal{C}'$ contains at most $2t$ vertices of C. Consequently, C^* contains at least $|C| - 2t$ vertices of C. ◀

Based on this lemma, we can now show that there are only linearly many options for the largest cluster of any optimal clustering, if the core has size at least $2t^2 + 4t + 1$.

▶ **Lemma 4.4.** *Let $G = (V, E)$ be a split graph with core C and periphery P, let $\omega : E \to [1, t]$ be an edge-weight function, and let $\mathcal{C}$ be an optimal clustering for G and ω. If $|C| \geq 2t^2+4t+1$, then either $\mathcal{C}$ contains the cluster C or there is some periphery vertex $v \in P$ with degree at least $|C| - 2t$ such that $\mathcal{C}$ contains the cluster $N[v]$.*

Proof. Assume towards a contradiction that the statement does not hold. Then, $\mathcal{C}$ contains neither the cluster C nor the cluster $N[v]$ for any periphery vertex $v \in P$ with degree at least $|C| - 2t$. Let C^* be a cluster of $\mathcal{C}$ with the most vertices of C. Since C has size at least $6t + 1$ and $\mathcal{C}$ is an optimal clustering for G and ω, Lemma 4.3 implies that C^* contains at least $|C| - 2t \geq 2t^2 + 1$ vertices of C.

Let v^* be the unique periphery vertex of C^* if such a vertex exists and an arbitrary vertex of C^*, otherwise. In both cases, there is a vertex $v \in (N(v^*) \cap C) \setminus C^*$, since C is not a cluster in $\mathcal{C}$ and for no periphery vertex $w \in P$ with degree at least $|C| - 2t$, $N[w]$ is a cluster of $\mathcal{C}$. We show that we can obtain a better clustering by moving vertex v to C^*. Note that $C^* \cup \{v\}$ is a clique in G, since v is adjacent to v^* and all vertices of $(C^* \setminus \{v^*\}) \cup \{v\}$ are from C. Let $\mathcal{C}'$ be the clustering for G obtained from $\mathcal{C}$ by moving vertex v to C^*. We show that $\mathcal{C}'$ is a better clustering for G and ω than $\mathcal{C}$. To this end, we analyze the total weight incident with vertex v under both $\mathcal{C}$ and $\mathcal{C}'$. Let C_v be the cluster of $\mathcal{C}$ containing v. Since $\mathcal{C} \setminus \{C^*\}$ contains at most $2t$ vertices of C, C_v has size at most $2t + 1$. Hence, v is incident with at most $2t$ edges in $E(\mathcal{C})$, each of weight at most t. Moreover, v is incident with $|C^*| \geq 2t^2 + 1$ edges of weight at least one each in $E(\mathcal{C}')$. Hence, $\mathcal{C}'$ is a better clustering for G and ω than $\mathcal{C}$. This contradicts the fact that $\mathcal{C}$ is an optimal clustering for G and ω. ◀

With this property at hand, we are now able to show that WCD can be solved in $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time on split graphs with a small or a very large core.

▶ **Lemma 4.5.** *Let $G = (V, E)$ be a split graph with core C and let $\omega : E \to [1, t]$ be an edge-weight function. One can find an optimal clustering for G and ω in $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time if $|C| \leq 6t$ or $|C| \geq 2t^2 + 4t + 1$.*

Proof. If $|C| \leq 6t$, then we can find an optimal clustering for G and ω in $3^{|C|} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time due to Lemma 4.1.

Otherwise, if $|C| \geq 2t^2 + 4t + 1$, Lemma 4.4 implies that each optimal clustering for G and ω contains a cluster C^* such that $C^* = C$ or $C^* = N[v]$ for some periphery vertex $v \in P$ with degree at least $|C| - 2t$. Since these are at most $|P| + 1 \in \mathcal{O}(n)$ possibilities, we can perform an initial branching for the choice of C^*. For each such choice for C^*, we find an optimal clustering $\mathcal{C}^*$ for $G - C^*$ and ω and return the best clustering $\mathcal{C}^* \cup \{C^*\}$ over all choices of C^*. Note that this algorithm is correct due to Lemma 4.4. The initial branching can be done in $n^{\mathcal{O}(1)}$ time and for each branching-instance we can find an optimal solution for $G - C^*$ and ω in $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time, since $G - C^*$ is a split graph with a core $C \setminus C^*$ of size at most $2t$.

Hence, in both cases, we find an optimal solution for I in the desired running time. ◀

Thus, to obtain an algorithm for WCD on split graphs, we now show how to solve the case that the core has size at least $6t + 1$ and at most $2t^2 + 4t$ and use this to bound the total running time.

▶ **Theorem 4.6.** WCD *can be solved in $t^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time on split graphs, where t denotes the largest edge weight.*

Proof. Let $I = (G = (V, E), \omega : E \to [1, t], k)$ be an instance of WCD, where G is a split graph with core C and periphery P. We show how to obtain an optimal clustering for G and ω in time $t^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$.

By Lemma 4.5, we can achieve this running time if $|C| \leq 6t$ or $|C| \geq 2t^2 + 4t + 1$. Hence, in the following, we assume that $6t + 1 \leq |C| \leq 2t^2 + 4t$. Due to Lemma 4.3, this implies that each optimal clustering for G and ω contains a cluster C^* such that $|C^* \cap C| \geq |C| - 2t$. Based on this fact, we can find an optimal clustering for G and ω by iterating over all $\widehat{C} \subseteq C$ of size at least $|C| - 2t$ and finding for each cluster $C^* \in \{\widehat{C}\} \cup \{\widehat{C} \cup \{v\} \mid v \in P, \widehat{C} \subseteq N(v)\}$ an optimal clustering for $G - C^*$ and ω. Over all such choices of C^*, we return the best clustering $\{C^*\} \cup \mathcal{C}^*$, where $\mathcal{C}^*$ is an optimal clustering for $G - C^*$. Due to Lemma 4.3, this algorithm finds an optimal clustering for G and ω.

It remains to show the running time. Since there are at most $|C|^{2t} = (2t^2+4t+1)^{2t} \in t^{\mathcal{O}(t)}$ distinct subsets $\widehat{C}$ of C with $|\widehat{C}| \geq |C| - 2t$ and for each such subset we consider at most $|P| + 1 \in \mathcal{O}(n)$ possible clusters C^*, we have to find an optimal clustering for at most $t^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ subgraphs $G - C^*$ of G. For each choice for C^*, $G - C^*$ is a split graph with a core of size $|C \setminus C^*| \leq 2t$. We can thus find an optimal clustering for $G - C^*$ in $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ time due to Lemma 4.5. Hence, the total running time is $t^{\mathcal{O}(t)} \cdot 2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)} = t^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$. ◀

On dense split graphs, we obtain an even faster algorithm.

▶ **Theorem 4.7.** WCD *can be solved in* $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ *time on dense split graphs, where* t *denotes the largest edge weight.*

Proof. Let $I = (G = (V, E), \omega : E \to [1, t], k)$ be an instance of WCD, where G is a dense split graph with core C and periphery P. We show how to obtain an optimal clustering for G and ω in time $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$.

Due to Lemma 4.5, we can achieve this running time if $|C| \leq 6t$. Hence, assume in the following, that $|C| > 6t$. Let $\mathcal{C}$ be an optimal clustering for G and ω. Due to Lemma 4.3, $\mathcal{C}$ contains a cluster C^* such that $|C^* \cap C| \geq |C| - 2t > 4t$. We show that $C^* := N[p] = \{p\} \cup C$ for some periphery vertex p. To this end, we first show that C^* contains all core vertices. Let $C' := C \setminus C^*$ denote the core vertices that are not in C^*. Since G is a dense split graph, each vertex of C' is adjacent to each vertex of C^*. Hence, moving all vertices of C' to the cluster C^* yields a valid clustering $\mathcal{C}'$. If $C' \neq \emptyset$, this clustering improves over $\mathcal{C}$, since each vertex of C' was adjacent to at most one periphery vertex, which implies that

$$\omega(E(\mathcal{C}) \setminus E(\mathcal{C}')) \leq |C'| \cdot t < |C'| \cdot 4t \leq |C'| \cdot |C^*| \leq \omega(E(\mathcal{C}') \setminus E(\mathcal{C})).$$

Since $\mathcal{C}$ is an optimal clustering for G and ω, this implies that $C' = \emptyset$ and thus C^* contains all core vertices. Moreover, due to the optimality of $\mathcal{C}$, C^* contains one periphery vertex, as otherwise adding an arbitrary periphery vertex to C^* would yield a better valid clustering, since G is a dense split graph. Hence, $C^* = N[p] = C \cup \{p\}$ for some periphery vertex $p \in P$. This implies that C^* is the only cluster of $\mathcal{C}$ that contains any edges, which further implies that $\omega(\mathcal{C}) = \omega(E(C^*)) = \omega(E(N[p]))$.

Hence, to find an optimal clustering for G and ω it suffices to find a periphery vertex $p \in P$ that maximizes $\omega(E(N[p]))$. This can be done in polynomial time. ◀

The result also gives a dichotomy with respect to the neighborhood diversity: Theorem 3.6 showed NP-hardness for neighborhood diversity 3 and $t = 2$. We now show an FPT-algorithm with respect to t for neighborhood diversity at most 2. Note that the graphs with neighborhood diversity at most 2 are a subset of the cographs but not a subset of the split graphs, since complete bipartite graphs have neighborhood diversity 2.

▶ **Corollary 4.8.** *On graphs with neighborhood diversity at most 2,* WCD *can be solved in* $2^{\mathcal{O}(t)} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. Note that if a graph is already a cluster graph, an optimal clustering of that graph simply contains all edges. The only graphs of neighborhood diversity at most 2 that are not cluster graphs are complete bipartite graphs and dense split graphs. On bipartite graphs, WCD asks simply for a maximum weight matching which can be solved in polynomial time [24]. For dense split graphs, Theorem 4.7 implies that we can solve WCD in the stated running time. ◀

5 Further FPT-Algorithms

Bounded-Treewidth Modulators to Clique. In the following, we extend ideas of the FPT-algorithm for split graphs to a generalization of split graphs. Namely, we show that WCD can be solved in FPT-time when parameterized by $t+r$ on graphs when a treewidth-r clique modulator is provided. Here, a *clique modulator* in a graph G is a vertex set M such that $G-M$ is a complete graph. The treewidth of a clique modulator M is defined to be the treewidth of $G[M]$. We now consider the parameter treewidth r of a given clique modulator of G and show that WCD admits an FPT-algorithm for this parameter.

▶ **Lemma 5.1 (*).** *Let G be a graph with edge-weight function $\omega : E \to [1,t]$ and let M be a treewidth-r modulator in G to a clique C. Then, if $|C| \geq 2\cdot(t\cdot(r+1))^2+4t\cdot(r+1)+1$, any optimal clustering $\mathcal{C}$ for G and ω contains the cluster C or there is some clique $K \subseteq M$ such that $\mathcal{C}$ contains the cluster $K \cup (N^{\cap}(K) \cap C)$ and this cluster contains at least $|C|-2(t\cdot(r+1))$ vertices of C.*

Note that this statement is a direct generalization of Lemma 4.4, since the periphery of a split graph G is a treewidth-0 modulator to the core of G.

With this lemma at hand, we are now able to present an algorithm for WCD when parameterized by t and the treewidth of a given clique modulator.

▶ **Theorem 5.2.** WCD *can be solved in time $2^{\mathcal{O}(t^2\cdot r^2)}\cdot n^{\mathcal{O}(1)}$ when given a treewidth-r clique modulator M for the input graph G.*

Proof. First, assume that $C := V \setminus M$ has size at most $2(t\cdot(r+1))^2+4t\cdot(r+1)$ then the input graph has treewidth $\mathcal{O}(t^2\cdot r^2)$ and we can solve WCD on this graph in time $2^{\mathcal{O}(t^2\cdot r^2)}\cdot n^{\mathcal{O}(1)}$ [25].[3]

Otherwise, C has size at least $2(t\cdot(r+1))^2+4t\cdot(r+1)+1$. Then, by Lemma 5.1, for each optimal clustering $\mathcal{C}$, either (a) $\mathcal{C}$ contains C or (b) there is a clique K in $G[M]$ such that $\mathcal{C}$ has a cluster consisting of K plus all of the (at least $|C|-2(t\cdot(r+1))$ many) common neighbors of K in C. Now observe that in Case (a) we may simply remove C from G and compute an optimal clustering for $G-C=G[M]$ which has treewidth at most r. This can be done in $2^{\mathcal{O}(r)}\cdot n^{\mathcal{O}(1)}$ time [25]. Otherwise, in Case (b), since $G[M]$ has treewidth at most r, we can enumerate all cliques K of $G[M]$ in $2^{\mathcal{O}(r)}\cdot n^{\mathcal{O}(1)}$ time, for example using the fact that $G[M]$ has degeneracy at most r. Now, for each clique K, we consider the case that the optimal clustering contains a cluster C' consisting of K and of all common neighbors of K in C. To compute the optimal clustering in that case, we may remove C' and find an optimal clustering for the remaining graph $G-C'=G[M\cup C\setminus C']$. This graph has treewidth at most $2(t\cdot(r+1))+r$ since $|C\setminus C'|\leq 2(t\cdot(r+1))$ and $G[M]$ has treewidth r. Thus, an optimal clustering for this graph can be computed in time $2^{\mathcal{O}(t\cdot r)}\cdot n^{\mathcal{O}(1)}$. The total running time for Case (b) is thus $2^r\cdot n^{\mathcal{O}(1)}\cdot 2^{\mathcal{O}(t\cdot r)}\cdot n^{\mathcal{O}(1)}$. ◀

Parameterization by (Split) Cluster Vertex Deletion Number. Next, we focus on the cluster vertex deletion number cvd of the input graph, that is, the minimum number of vertices to remove from G to obtain a cluster graph. Recently, it was shown that CLUSTER DELETION admits an FPT-algorithm when parameterized by cvd [14]. In contrast, WCD

[3] The algorithm described in [25] solves the unweighted problem via dynamic programming on tree decompositions; it can be easily adapted to the weighted case by summing up over edge weights instead of counting edges inside clusters.

is known to be NP-hard even on graphs with cvd = 2 [19]. This motivates the study of the combined parameter cvd and the maximum edge weight t. We show an FPT-algorithm for WCD parameterized by this combined parameter.

To this end, we first provide some additional notation. Let $G = (V, E)$ be a graph. For a cluster modulator M, let $\mathcal{B}(M)$ be the collection of connected components of $G - M$, that is, the clusters of the cluster graph $G - M$. The individual clusters of $\mathcal{B}(M)$ are referred to as *bags*. If the cluster modulator is clear from the context, we may also only write $\mathcal{B}$.

▶ **Theorem 5.3.** WCD *can be solved in* $(t \cdot \mathrm{cvd})^{\mathcal{O}(t \cdot \mathrm{cvd})} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. Let $I := (G := (V, E), \omega, k)$ be an instance of WCD with $\omega : E \to [1, t]$. The algorithm consists of two steps. First, we compute a minimum-size cluster modulator M for G. Second, we iterate over all possible clusterings of $G[M]$ and compute for each such clustering $\mathcal{C}_M$ the best clustering of G that extends $\mathcal{C}_M$. To solve the latter task, we use dynamic programming over subsets to find an optimal way to distribute the vertices of the bags of $\mathcal{B}(M)$ among the clusters of $\mathcal{C}_M$.

Let M be a minimum-size cluster modulator for G with collection of bags $\mathcal{B}$ and let $\mathcal{C}_M$ be a fixed clustering of $G[M]$. We fix an arbitrary ordering of the bags of $\mathcal{B}$ and let B_i denote the ith bag according to this fixed ordering. The dynamic programming table T has entries of type $T[i, \mathcal{C}'_M]$ with $i \in [0, |\mathcal{B}|]$ and $\mathcal{C}'_M \subseteq \mathcal{C}_M$ and stores the total edge weight of an optimal way to distribute the vertices of the first i bags among the clusters of $\mathcal{C}'_M$. Hence, the base case for $i = 0$ and each $\mathcal{C}'_M \subseteq \mathcal{C}_M$ is defined as $T[0, \mathcal{C}'_M] := \omega(\mathcal{C}'_M)$. The key observation for the dynamic program is the fact that no cluster in any clustering for G can contain vertices of more than one bag. Hence, to find an optimal way to distribute the vertices of the first i bags among the clusters of $\mathcal{C}'_M$ it is sufficient to check for each subset $\mathcal{C}''_M \subseteq \mathcal{C}'_M$ for an optimal distribution of the vertices of the ith bag among the clusters of $\mathcal{C}''_M$ and combine it with an optimal way to distribute the vertices of the first $i - 1$ bags among the clusters of $\mathcal{C}'_M \setminus \mathcal{C}''_M$. This leads to the following recurrence, where $\mathrm{OPT}(B_i, \mathcal{C}''_M)$ denotes an optimal way to distribute the vertices of the ith bag among the clusters of $\mathcal{C}''_M$:

$$T[i, \mathcal{C}'_M] := \max_{\mathcal{C}''_M \subseteq \mathcal{C}'_M} \mathrm{OPT}(B_i, \mathcal{C}''_M) + T[i - 1, \mathcal{C}'_M \setminus \mathcal{C}''_M].$$

By the above argumentation, this recurrence is correct. Hence, a total weight of a best clustering for G that extends $\mathcal{C}_M$ is stored in $T[|\mathcal{B}|, \mathcal{C}_M]$. Moreover, a corresponding clustering can be found via trace back.

It thus remains to show the running time of the dynamic program. To this end, we first need to show that the value $\mathrm{OPT}(B_i, \mathcal{C}''_M)$ for each $i \in [1, |\mathcal{B}|]$ and each $\mathcal{C}''_M \subseteq \mathcal{C}_M$ can be computed in the desired running time with respect to $\mathrm{cvd} + t$.

Let $G_i := G[B_i \cup \bigcup_{C' \in \mathcal{C}''_M}]$. Due to Observation 2.2, we can assume without loss of generality that $G_i[M]$ is a cluster graph. Hence, G_i is unipolar, since $G_i - M$ is a clique on the vertices of B_i. Let (G'_i, k') be the graph and edge-weight function obtained from performing the clustering-merge of $\mathcal{C}''_M$ for G_i and ω. Note that G'_i is a split graph and the largest assigned weight by ω' is at most $|M| \cdot t = \mathrm{cvd} \cdot t$. Hence, we can find an optimal clustering $\mathcal{C}'_i$ for G'_i and ω' in time $(\mathrm{cvd} \cdot t)^{\mathcal{O}(\mathrm{cvd} \cdot t)} \cdot n^{\mathcal{O}(1)}$ due to Theorem 4.6. By Lemma 2.4, we can then find a best clustering for G_i and ω that extends $\mathcal{C}''_M$ in polynomial time. Hence, for each $i \in [1, |\mathcal{B}|]$ and each $\mathcal{C}''_M \subseteq \mathcal{C}_M$, the value $\mathrm{OPT}(B_i, \mathcal{C}''_M)$ can be computed in $(\mathrm{cvd} \cdot t)^{\mathcal{O}(\mathrm{cvd} \cdot t)} \cdot n^{\mathcal{O}(1)}$ time.

Concluding, the whole algorithm runs in the desired running time, since (a) M can be computed in $1.811^{\mathrm{cvd}} \cdot n^{\mathcal{O}(1)}$ time [27], (b) all clusterings for $G[M]$ can be enumerated in $|M|^{|M|} = \mathrm{cvd}^{\mathrm{cvd}}$ time, (c) for each clustering $\mathcal{C}_M$ for $G[M]$, the dynamic programming table T has $2^{|\mathcal{C}_M|} \cdot n \leq 2^{\mathrm{cvd}} \cdot n$ entries, and (d) each such entry can be computed in $(\mathrm{cvd} \cdot t)^{\mathcal{O}(\mathrm{cvd} \cdot t)} \cdot n^{\mathcal{O}(1)}$ time. ◀

We can extend the idea of the above algorithm to the even smaller parameter split cluster vertex deletion number scvd, that is the minimum number of vertices to remove from G such that in the remaining graph each connected component is a split graph. More formally, let $G = (V, E)$ be a graph. We say that a vertex set $M \subseteq V$ is a *split cluster modulator for G* if $G - M$ is a split cluster graph. The *split cluster vertex deletion number* scvd of G is defined as the size of the smallest split cluster modulator for G. For a split cluster modulator M, let $\mathcal{B}(M)$ be the collection of connected components of $G - M$, that is, the maximal induced split graphs of the split cluster graph $G - M$. The individual maximal induced split graphs of $\mathcal{B}(M)$ are referred to as *bags*. If the split cluster modulator is clear from the context, we may also only write $\mathcal{B}$.

Note that scvd is a smaller parameter than cvd, since each cluster graph is also a split cluster graph. We now show that WCD can be solved in FPT-time on general graphs when parameterized by the largest edge weight t and scvd. To this end, we first give an algorithm for computing a clustering when given a vertex set S such that $G - S$ is a split graph with core C and periphery P. In that case, $S \cup P$ is a treewidth-$|S|$ modulator to the core of $G - S$ which is a clique. Thus, Theorem 5.2 implies the following.

▶ **Corollary 5.4.** *Let $G = (V, E)$ be a graph and let $\omega : E \to [1, t]$ be an edge-weight function. Let $S \subseteq V$ such that $G - S$ is a split graph. One can find an optimal clustering for G and ω in $2^{\mathcal{O}((t \cdot |S|)^2)} \cdot n^{\mathcal{O}(1)}$ time.*

We now use this algorithm as a subroutine in an algorithm similar to the one of Theorem 5.3.

▶ **Theorem 5.5.** WCD *can be solved in $2^{\mathcal{O}(\mathrm{scvd}^2 \cdot t^2)} \cdot n^{\mathcal{O}(1)}$ time.*

Proof. After computing a smallest split cluster modulator M in $2^{\mathcal{O}(\mathrm{scvd})} \cdot n^{\mathcal{O}(1)}$ time [6], we perform the same dynamic program as in the proof of Theorem 5.3 over the bags resulting from the modulator M. Note that each bag is now a split graph instead of a clique. The only difference then lies in computing the value $\mathrm{OPT}(B_i, \mathcal{C}''_M)$ for each bag B_i and each $\mathcal{C}''_M \subseteq \mathcal{C}_M$. Due to Corollary 5.4, this can be done in $2^{\mathcal{O}((t \cdot |M|)^2)} \cdot n^{\mathcal{O}(1)}$ time. Hence, the whole algorithm also runs in $2^{\mathcal{O}((t \cdot |M|)^2)} \cdot n^{\mathcal{O}(1)}$ time. ◀

6 Kernelization lower bounds

Given the FPT-algorithms for parameter t presented in Section 4, a natural next question is to ask for a polynomial kernel for t. In this section, we show that WCD on (dense) split graphs does not admit a polynomial kernel when parameterized by $t + \mathrm{vc}$, unless NP $\subseteq$ coNP/poly. Moreover, we show that even a polynomial Turing kernel for WCD on (dense) split graphs when parameterized by $t+\mathrm{vc}$ is unlikely, by showing that the problem is WK[1]-hard. Roughly speaking, WK[1]-hardness for a parameterized problem means that a polynomial Turing kernel for this problem is unlikely [18].

To show our kernelization lower bounds, we present a chain of polynomial parameter transformations starting from Set Cover when parameterized by the size of the universe.

Set Cover
Input: A set X, a collection $\mathcal{F}$ of subsets of X, and an integer k.
Question: Is there a subset $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k, such that every element of X occurs in at least one set of $\mathcal{F}'$?

Since Set Cover when parameterized by the size of the universe, is WK[1]-hard [18] and does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly [9], this then implies that WCD on (dense) split graphs when parameterized by $t + \mathrm{vc}$ is WK[1]-hard and does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly.

▶ **Lemma 6.1** ([18]). *There is a polynomial parameter transformation from* Set Cover *parameterized by the size of the universe to* Exact Cover *parameterized by the size of the universe.*

This statement follows, since both Set Cover and Exact Cover parameterized by the size of the universe are WK[1]-complete, and the class of WK[1]-complete problems is closed under polynomial parameter transformations [18].

We now present a polynomial parameter transformation from Exact Cover to Uniform Exact Cover, both parameterized by size of the universe.

▶ **Proposition 6.2** (*). *There is a polynomial parameter transformation from* Exact Cover *parameterized by the size of the universe to* Uniform Exact Cover *parameterized by the size of the universe.*

Based on the observed polynomial parameter transformation from Exact Cover parameterized by size of the universe to WCD on dense split graph when parameterized by $t + \mathrm{vc}$ in Proposition 3.2, we conclude the following due to the chain of polynomial parameter transformations (Lemma 6.1, Proposition 6.2, and Proposition 3.2) and the kernelization lower bounds for Set Cover.

▶ **Theorem 6.3.** WCD *on dense split graphs when parameterized by* $t + \mathrm{vc}$ *does not admit a polynomial kernel, unless* $\mathrm{NP} \subseteq \mathrm{coNP/poly}$*. Moreover,* WCD *on dense split graphs is* WK[1]*-hard when parameterized by* $t + \mathrm{vc}$*.*

Additionally, due to Lemma 6.1, Proposition 6.2, and the kernelization lower bounds for Set Cover, we also derive the following.

▶ **Corollary 6.4.** Uniform Exact Cover *when parameterized by the size of the universe does not admit a polynomial kernel, unless* $\mathrm{NP} \subseteq \mathrm{coNP/poly}$*. Moreover,* Uniform Exact Cover *is* WK[1]*-hard when parameterized by the size of the universe.*

7 Conclusion

The most immediate open question is whether Weighted Cluster Deletion is polynomial-time solvable for constant values of t when the input graph is a proper interval graph. Another direction would be to improve the running time for the FPT-algorithm on split graphs to close the gap between the upper and lower bound. Finally it would be interesting to consider other NP-hard edge-weighted problems using the parameter t. A good candidate would be Cluster Editing [1, 26] where the task is to obtain a cluster graph by modifying as few edges as possible. The weighted version of this problem has also received a lot of attention over the years [2, 7]. As for Weighted Cluster Deletion, such a study would need to focus on graph classes where unweighted Cluster Editing is polynomial-time solvable. Another direction could be to consider problems related to Cluster Deletion, where each cluster is demanded to fulfill some relaxed cluster definition, for example being a so-called s-plex [17].

References

1 Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *Proceedings of the 9th Conference on Computability in Europe (CiE '13)*, volume 7921 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2013. doi:10.1007/978-3-642-39053-1_5.

2 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. doi:10.1016/J.TCS.2009.05.006.

3 Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. doi:10.1016/J.IPL.2011.05.003.

4 Flavia Bonomo, Guillermo Durán, Amedeo Napoli, and Mario Valencia-Pabon. A one-to-one correspondence between potential solutions of the cluster deletion problem and the minimum sum coloring problem, and its application to P_4-sparse graphs. *Information Processing Letters*, 115(6-8):600–603, 2015. doi:10.1016/J.IPL.2015.02.007.

5 Flavia Bonomo, Guillermo Durán, and Mario Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015. doi:10.1016/j.tcs.2015.07.001.

6 Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10:16, 2015. doi:10.1186/S13015-015-0043-7.

7 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012. doi:10.1007/S00453-011-9595-1.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

10 Rodney G. Downey and Michael Ralph Fellows. *Fundamentals of Parameterized Complexity*. Springer Science & Business Media, 2013.

11 Elaine M. Eschen and Xiaoqiang Wang. Algorithms for unipolar and generalized split graphs. *Discrete Applied Mathematics*, 162:195–201, 2014. doi:10.1016/J.DAM.2013.08.011.

12 Wen-Yu Gao and Hang Gao. $2k$-vertex kernels for cluster deletion and strong triadic closure. *Journal of Computer Science and Technology*, 38:1431–1439, 2023. doi:10.1007/s11390-023-1420-1.

13 Yong Gao, Donovan R. Hare, and James Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013. doi:10.1016/j.disc.2013.08.017.

14 Jaroslav Garvardt, Nils Morawietz, André Nichterlein, and Mathias Weller. Graph clustering problems under the lens of parameterized local search. In Neeldhara Misra and Magnus Wahlström, editors, *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC '23)*, volume 285 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.20.

15 Petr A. Golovach, Pinar Heggernes, Athanasios L. Konstantinidis, Paloma T. Lima, and Charis Papadopoulos. Parameterized aspects of strong subgraph closure. *Algorithmica*, 82(7):2006–2038, 2020. doi:10.1007/S00453-020-00684-9.

16 Niels Grüttemeier and Christian Komusiewicz. On the relation of strong triadic closure and cluster deletion. *Algorithmica*, 82(4):853–880, 2020. doi:10.1007/S00453-019-00617-1.

17 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. doi:10.1137/090767285.

18 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/S00453-014-9910-8.

19 Giuseppe F. Italiano, Athanasios L. Konstantinidis, and Charis Papadopoulos. Structural parameterization of cluster deletion. In Chun-Cheng Lin, Bertrand M. T. Lin, and Giuseppe Liotta, editors, *Proceedings of the 17th International Conference and Workshops on Algorithms and Computation (WALCOM '23)*, volume 13973 of *Lecture Notes in Computer Science*, pages 371–383. Springer, 2023. doi:10.1007/978-3-031-27051-2_31.

20 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM J. Discret. Math.*, 30(1):343–366, 2016. doi:10.1137/140952636.

21 Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Deconstructing intractability – A multivariate complexity analysis of interval constrained coloring. *Journal of Discrete Algorithms*, 9(1):137–151, 2011. doi:10.1016/J.JDA.2010.07.003.

22 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/J.DAM.2012.05.019.

23 Athanasios L. Konstantinidis and Charis Papadopoulos. Cluster deletion on interval graphs and split related graphs. *Algorithmica*, 83(7):2018–2046, 2021. doi:10.1007/s00453-021-00817-8.

24 Harold W. Kuhn. The Hungarian Method for the Assignment Problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi:10.1007/978-3-540-68279-0_2.

25 Sebastian Ochs. Cluster deletion on unit disk graphs. Master's thesis, Philipps-Universität Marburg, 2023. URL: https://www.fmi.uni-jena.de/fmi_femedia/fakultaet/institute-und-abteilungen/informatik/algorithm-engineering/master-thesis-sebastian-ochs.pdf.

26 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. doi:10.1016/J.DAM.2004.01.007.

27 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65(2):323–343, 2021. doi:10.1007/s00224-020-10005-w.

28 Dekel Tsur. Cluster deletion revisited. *Information Processing Letters*, 173:106171, 2022. doi:10.1016/J.IPL.2021.106171.

Robust Bichromatic Classification Using Two Lines

Erwin Glazenburg
Utrecht University, The Netherlands

Thijs van der Horst
Utrecht University, The Netherlands
TU Eindhoven, The Netherlands

Tom Peters
TU Eindhoven, The Netherlands

Bettina Speckmann
TU Eindhoven, The Netherlands

Frank Staals
Utrecht University, The Netherlands

Abstract

Given two sets R and B of n points in the plane, we present efficient algorithms to find a two-line linear classifier that best separates the "red" points in R from the "blue" points in B and is robust to outliers. More precisely, we find a region $\mathcal{W}_B$ bounded by two lines, so either a halfplane, strip, wedge, or double wedge, containing (most of) the blue points B, and few red points. Our running times vary between optimal $O(n \log n)$ up to around $O(n^3)$, depending on the type of region $\mathcal{W}_B$ and whether we wish to minimize only red outliers, only blue outliers, or both.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Geometric Algorithms, Separating Line, Classification, Bichromatic, Duality

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.33

Related Version *Full Version*: `https://arxiv.org/abs/2401.02897` [14]

1 Introduction

Let R and B be two sets of at most n points in the plane. Our goal is to best separate the "red" points R from the "blue" points B using at most two lines. That is, we wish to find a region $\mathcal{W}_B$ bounded by lines ℓ_1 and ℓ_2 containing (most of) the blue points B so that the number of points k_R from R in the interior $\text{int}(\mathcal{W}_B)$ of $\mathcal{W}_B$ and/or the number of points k_B from B in the interior of the region $\mathcal{W}_R = \mathbb{R}^2 \setminus \mathcal{W}_B$ is minimized. We refer to these subsets $\mathcal{E}_R = R \cap \text{int}(\mathcal{W}_B)$ and $\mathcal{E}_B = B \cap \text{int}(\mathcal{W}_R)$ as the red and blue outliers, respectively, and define $\mathcal{E} = \mathcal{E}_R \cup \mathcal{E}_B$ and $k = k_R + k_B$.

The region $\mathcal{W}_B$ is either: (i) a halfplane, (ii) a *strip* bounded by two parallel lines ℓ_1 and ℓ_2, (iii) a *wedge*, i.e., one of the four regions induced by a pair of intersecting lines ℓ_1 and ℓ_2, or (iv) a *double wedge*, i.e., two opposing regions induced by a pair of intersecting lines

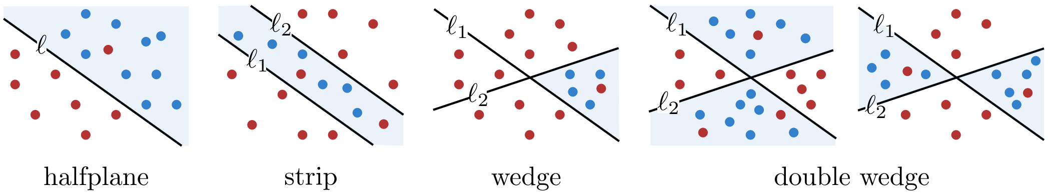


Figure 1 We consider separating R and B by at most two lines. This gives rise to four types of regions $\mathcal{W}_B$: halfplanes, strips, wedges, and two types of double wedges: hourglasses and bowties.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 33; pp. 33:1–33:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

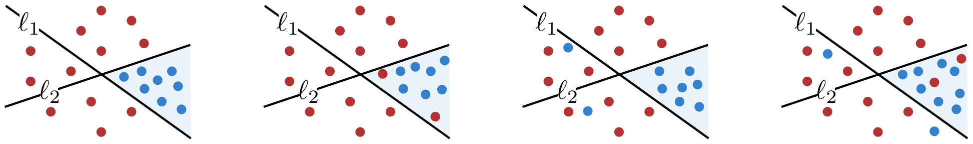
ℓ_1 and ℓ_2 (we further distinguish *hourglass* double wedges, that contain a vertical line, and the remaining *bowtie* double wedges), see Figure 1. We can reduce the case that $\mathcal{W}_B$ would consist of three regions to the single-wedge case by recoloring the points. For each of these cases for the shape of $\mathcal{W}_B$ we consider three problems: allowing only red outliers ($k_B = 0$) and minimizing k_R, allowing only blue outliers ($k_R = 0$) and minimizing k_B, or allowing both outliers and minimizing $k = k_R + k_B$. We present efficient algorithms for each of these problems, as shown in Table 1.

Motivation and related work. Classification is a key problem in computer science. The input is a labeled set of points and the goal is to obtain a procedure that, given an unlabeled point, assigns it a label that "fits it best", considering the labeled points. Classification has many direct applications, e.g. identifying SPAM in email messages, or tagging fraudulent transactions [20, 22], but is also the key subroutine in other problems such as clustering [1].

We restrict our attention to binary classification where our input is a set R of red points and a set B of blue points. We can compute whether R and B can be perfectly separated by a line (and compute such a line if it exists) in $O(n)$ time using linear programming. This extends to finding a separating hyperplane in case of points in $\mathbb{R}^d$, for some constant d [19].

Clearly, it is not always possible to find a hyperplane that perfectly separates the red and the blue points, see for example Figure 2, in which the blue points are actually all contained in a wedge. Hurtato et al. [16, 17] consider separating R and B in $\mathbb{R}^2$ using at most two lines ℓ_1 and ℓ_2. In this case, linear programming is unfortunately no longer applicable. Instead, they present $O(n \log n)$ time algorithms to compute a perfect separator (i.e., a strip, wedge, or double wedge containing all blue points but no red points), if it exists. These results were shown to be optimal [5], and can be extended to the case where B and R contain other geometric objects such as segments or circles, or to include constraints on the slopes [16]. Similarly, Hurtado et al. [18] considered similar strip and wedge separability problems for points in $\mathbb{R}^3$. Arkin et al. [4] show how to compute a 2-level binary space partition (a line ℓ and two rays starting on ℓ) separating R and B in $O(n^2)$ time, and a minimum height h-level tree, with $h \leq \log n$, in $n^{O(\log n)}$ time. Even today, computing perfect bichromatic separators with particular geometric properties remains an active research topic [2].

Alternatively, one can consider separation with a (hyper-)plane but allow for outliers. Chan [8] presented algorithms for linear programming in $\mathbb{R}^2$ and $\mathbb{R}^3$ that allow for up to k violations –and thus solve hyperplane separation with up to k outliers– that run in $O((n + k^2) \log n)$ and $O(n \log n + k^{11/4} n^{1/4} \operatorname{polylog} n)$ time, respectively. In higher (but constant) dimensions, only trivial solutions are known. For arbitrary (non-constant) dimensions the problem is NP-hard [3]. There is also a fair amount of work that aims to find a halfplane that minimizes some other error measure, e.g. the distance to the farthest misclassified point, or the sum of the distances to misclassified points [7, 15].



perfect

minimize k_R

minimize k_B

minimize k

Figure 2 Perfectly separating R and B may require more than one line. When considering outliers, we may allow '(and minimize) only red outliers, only blue outliers, or both.

Table 1 An overview of our results. Results shown in the full version [14] are marked with “full”. Expected running times are marked with a ‡.

region $\mathcal{W}_B$	minimize k_R		minimize k_B		minimize k	
halfplane	$O(n \log n)$	full	$O(n \log n)$	full	$O((n + k^2) \log n)$	[8]
strip	$O(n \log n)$	[21]	$O(n^2 \log n)$	full	$O(n^2 \log n)$	full
wedge	$O(n^2)$	[21]	$O(n^{5/2} \log n)$‡	§5.2		
	$O(n \log n)$	§5.1	$O(nk_B^2 \log^2 n \log k_B)$	§5.3	$O(nk^2 \log^3 n \log k)$	§5.3
double bowtie	$O(n^2)$	§6	$O(n^2 \log n)$	§6	$O(n^2 k \log^3 n \log k)$	§6

Separating points using more general non-hyperplane separators and outliers while incorporating guarantees on the number of outliers seems to be less well studied. Seara [21] showed how to compute a strip containing all blue points, while minimizing the number of red points in the strip in $O(n \log n)$ time. Similarly, he presented an $O(n^2)$ time algorithm for computing a wedge with the same properties. Armaselu and Daescu [6] show how to compute and maintain a smallest circle containing all red points and the minimum number of blue points. In this paper, we take some further steps toward the fundamental but challenging problem of computing a robust non-linear separator that provides performance guarantees.

Results. We present efficient algorithms for computing a region $\mathcal{W}_B = \mathcal{W}_B(\ell_1, \ell_2)$ defined by at most two lines ℓ_1 and ℓ_2 containing only the blue points, that are robust to outliers. Our results depend on the type of region $\mathcal{W}_B$ we are looking for, i.e., halfplane, strip, wedge, or double wedge, as well as on the type of outliers we allow: red outliers (counted by k_R), blue outliers (counted by k_B), or all outliers (counted by k). Refer to Table 1 for an overview.

Our main contributions are efficient algorithms for when $\mathcal{W}_B$ is really bounded by two lines. All these versions can be solved by a simple $O(n^4)$ time algorithm that explicitly considers all candidate regions. However, we show that we can do significantly better in every case.

In particular, in the versions where we minimize the number of red outliers k_R we achieve significant speedups. For example, we can compute an optimal wedge $\mathcal{W}_B$ containing B and minimizing k_R in optimal $\Theta(n \log n)$ time (which improves an earlier $O(n^2)$ time algorithm from Seara [21]). We use two types of duality transformations that allow us to map each point $p \in R \cup B$ into a *forbidden region* E_p in a low-dimensional parameter space, such that: *i)* every point s in this parameter space corresponds to a region $\mathcal{W}_B(s)$, and *ii)* this region $\mathcal{W}_B(s)$ misclassifies point p if and only if this point s lies in E_p. This allows us to solve the problem by computing a point that lies in the minimum number of forbidden regions.

Surprisingly, the versions of the problem in which we minimize the number of blue outliers k_B are much more challenging. For none of these versions we can match our running times for minimizing k_R, while needing more advanced tools. For example, for the single wedge version, we use dynamic lower envelopes to obtain a batched query problem that we solve using spanning-trees with low stabbing number [10]. See Section 5.2.

For the case where both red and blue outliers are allowed and we minimize k, we present output-sensitive algorithms whose running time depends on the optimal value of k. We essentially fix one of the lines ℓ_1, and use linear programming (LP) with violations [8, 13] to compute an optimal line ℓ_2 that together with ℓ_1 defines $\mathcal{W}_B$. We show that by using results on $\leq k$-levels, a recent data structure for dynamic LP with violations [13], and binary searching, we can achieve algorithms with running times around $O(n^2 k \operatorname{polylog} n)$.

Outline. We give some additional definitions and notation in Section 2, and in Section 3 we present a characterization of optimal solutions that lead to our simple $O(n^4)$ time algorithm for any type of wedges. In Sections 4, 5, and 6 we discuss the case when $\mathcal{W}_B$ is, respectively, a strip, wedge, or double wedge. In each of these sections we separately go over minimizing the number of red outliers k_R, the number of blue outliers k_B, and the total number of outliers k. We wrap up with some concluding remarks and future work in Section 7. Omitted proofs can be found in the full version [14].

2 Preliminaries

In this section we discuss some notation and concepts used throughout the paper. For ease of exposition we assume $B \cup R$ contains at least three points and is in general position, i.e., that all coordinate values are unique, and that no three points are colinear.

Notation. Let ℓ^- and ℓ^+ be the two halfplanes bounded by line ℓ, with ℓ^- below ℓ (or left of ℓ if ℓ is vertical). Any pair of lines ℓ_1 and ℓ_2, with the slope of ℓ_1 smaller than that of ℓ_2, subdivides the plane into at most four interior-disjoint regions: $\text{North}(\ell_1, \ell_2) = \ell_1^+ \cap \ell_2^+$, $\text{East}(\ell_1, \ell_2) = \ell_1^+ \cap \ell_2^-$, $\text{South}(\ell_1, \ell_2) = \ell_1^- \cap \ell_2^-$ and $\text{West}(\ell_1, \ell_2) = \ell_1^- \cap \ell_2^+$. When ℓ_1 and ℓ_2 are clear from the context we may simply write North to mean $\text{North}(\ell_1, \ell_2)$, etc. We assign each of these regions to either B or R, so that $\mathcal{W}_B = \mathcal{W}_B(\ell_1, \ell_2)$ and $\mathcal{W}_R = \mathcal{W}_R(\ell_1, \ell_2)$ are the union of some elements of $\{\text{North}, \text{East}, \text{South}, \text{West}\}$. In case ℓ_1 and ℓ_2 are parallel, we assume that ℓ_1 lies below ℓ_2, and thus $\mathcal{W}_B = \text{East}$.

Duality. We make frequent use of the standard point-line duality [11], where we map objects in *primal* space to objects in a *dual* space. In particular, a primal point $p = (a, b)$ is mapped to the dual line $p^* : y = ax - b$ and a primal line $\ell : y = ax + b$ is mapped to the dual point $\ell^* = (a, -b)$. If in the primal a point p lies above a line ℓ, then in the dual the line p^* lies below the point ℓ^*.

For a set of points P with duals $P^* = \{p^* \mid p \in P\}$, we are often interested in the *arrangement* $\mathcal{A}(P^*)$, i.e., the vertices, edges, and faces formed by the lines in P^*. Two unbounded faces of $\mathcal{A}(P^*)$ are *antipodal* if their unbounded edges have the same two supporting lines. Since every line contributes to four unbounded faces, there are $O(n)$ pairs of antipodal faces. We denote the upper envelope of P^*, i.e., the polygonal chain following the highest line in $\mathcal{A}(P^*)$, by $\mathcal{U}(P^*)$, and the lower envelope by $\mathcal{L}(P^*)$.

3 Properties of an optimal separator

Next, we prove some structural properties about the lines bounding the region $\mathcal{W}_B$ containing (most of) the blue points in B.

▶ **Lemma 3.1.** *For the strip classification problem there exists an optimum where one line goes through two points and the other through at least one point.*

Proof. Clearly, we can shrink an optimal strip $\mathcal{W}_B(\ell_1, \ell_2)$ so that both ℓ_1 and ℓ_2 contain a (blue) point, say b_1 and b_2, respectively. Now rotate ℓ_1 around b_1 and ℓ_2 around b_2 in counter-clockwise direction until either ℓ_1 or ℓ_2 contains a second point. ◀

▶ **Lemma 3.2.** *For any wedge classification problem there exists an optimum where both lines go through a blue and a red point.*

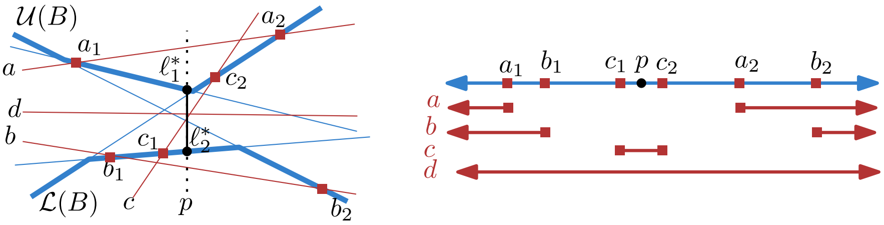


Figure 3 The four types of red lines and their forbidden region.

Proof sketch. Similar to the proof for strips, we show that any (double) wedge can be adjusted until both its lines go through a blue and a red point, without misclassifying any more points. Since this also holds for a given optimum, the lemma follows. ◀

A simple general algorithm. Lemma 3.2 tells us we only have to consider lines through red and blue points. Hence, there is a simple brute-force $O(n^4)$ time algorithm that considers all pairs of such lines, which works for both wedges and double wedges and any type of outliers. Refer to the full version for details.

4 Separation with a strip

In this section we consider the case where lines ℓ_1 and ℓ_2 are parallel, with ℓ_2 above ℓ_1, and thus $\mathcal{W}_B(\ell_1, \ell_2)$ forms a strip. We want B to be inside the strip, and R outside. We work in the dual, where we want to find two points ℓ_1^* and ℓ_2^* with the same x-coordinate such that vertical segment $\overline{\ell_1^*\ell_2^*}$ intersects the lines in B^* but not the lines in R^*. We briefly summarize our approach and our results.

Strip separation with red outliers. In this version, we wish to find a vertical line segment $\overline{\ell_1^*\ell_2^*}$ that intersects all lines in B^* and minimizes the number of lines from R^* it intersects. So we can assume that ℓ_1^* lies on the upper envelope $\mathcal{U}(B^*)$ of B^* and ℓ_2^* lies on the lower envelope $\mathcal{L}(B^*)$, since shortening $\overline{\ell_1^*\ell_2^*}$ can only decrease the number of red lines intersected. There is only one degree of freedom for choosing our segment, its x-coordinate, so our *parameter space* is $\mathbb{R}$. We parameterize $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ over $\mathbb{R}$, so that each point $p \in \mathbb{R}$ in the parameter space corresponds to the vertical segment $\overline{\ell_1^*\ell_2^*}$ on the line $x = p$. See Figure 3. We map every red line r to a forbidden region (an interval) in this parameter space, in which $\overline{\ell_1^*\ell_2^*}$ would intersect r. Our goal is then to compute a point p that is contained in the minimum number of such forbidden intervals. We can do so in $O(n \log n)$ time by sorting and scanning. This matches an existing result by Seara [21].

Strip separation with blue outliers. In this version, the vertical segment $\overline{\ell_1^*\ell_2^*}$ may intersect no red lines and as many blue lines as possible. That means that there is a $\overline{\ell_1^*\ell_2^*}$ that is a maximal vertical segment in a face of $\mathcal{A}(R^*)$. We sweep a vertical line ℓ over $\mathcal{A}(R^*)$ while, for each face F (defining a candidate segment $F \cap \ell$ for $\overline{\ell_1^*\ell_2^*}$) we maintain the number of blue lines that intersect the candidate segment. This leads to an $O(n^2 \log n)$ time algorithm for computing an optimal segment, and thus an optimal strip.

Strip separation with both outliers. In this version, the vertical segment $\overline{\ell_1^*\ell_2^*}$ may misclassify both red and blue lines, but as few as possible. There is much less structure for where an optimal segment can be than before, since an optimal segment can now intersect any

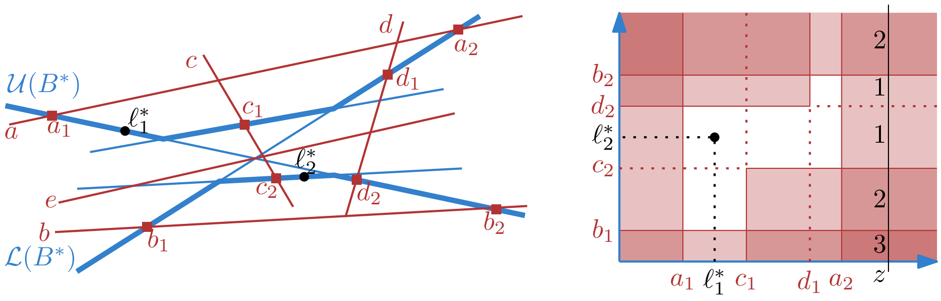


Figure 4 The arrangement of $B^* \cup R^*$ with its parameter space and forbidden regions.

number of blue or red lines. We sweep over the full arrangement $\mathcal{A}(R^* \cup B^*)$ with a vertical line. We maintain a datastructure that, given a point ℓ_2^* on the sweepline, can find a second point ℓ_1^* such that the number of outliers $|\mathcal{E}(\ell_1, \ell_2)|$ is minimized. Each time the sweepline crosses a vertex of $\mathcal{A}(R^* \cup B^*)$ we update the datastructure and perform one query, both in $O(\log n)$ time, resulting in an $O(n^2 \log n)$ algorithm.

▶ **Theorem 4.1.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a strip $\mathcal{W}_B$ minimizing (i) the number of red outliers k_R in $O(n \log n)$ time, (ii) the number of blue outliers k_B in $O(n^2 \log n)$ time, or (iii) the number of outliers k in $O(n^2 \log n)$ time.*

5 Separation with a wedge

We consider the case where the region $\mathcal{W}_B$ is a single wedge and $\mathcal{W}_R$ is the other three wedges. In Sections 5.1, 5.2, and 5.3 we show how to minimize k_R, k_B, and k, respectively.

5.1 Wedge separation with red outliers

We distinguish between $\mathcal{W}_B$ being an East or West wedge, and a North or South wedge. In either case we can compute optimal lines ℓ_1 and ℓ_2 defining $\mathcal{W}_B$ in $O(n \log n)$ time.

Finding an East or West wedge. We wish to find two lines ℓ_1 and ℓ_2 such that every blue point and as few red points as possible lie above ℓ_1 and below ℓ_2. In the dual this corresponds to points ℓ_1^* and ℓ_2^* such that all blue lines and as few red lines as possible lie below ℓ_1^* and above ℓ_2^*, as in Figure 4.

Clearly ℓ_1^* must lie above $\mathcal{U}(B^*)$, and ℓ_2^* below $\mathcal{L}(B^*)$, and again we can assume they lie on $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$, respectively. We now have two degrees of freedom, one for choosing ℓ_1^* and one for choosing ℓ_2^*. Again we parameterize $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$, but this time over $\mathbb{R}^2$, such that a point (p, q) in this parameter space corresponds to two dual points ℓ_1^* and ℓ_2^*, with ℓ_1^* on $\mathcal{U}(B^*)$ at $x = p$ and $\ell_2^*(y)$ on $\mathcal{L}(B^*)$ at $x = q$, as illustrated in Figure 4. We wish to find a value in our parameter space whose corresponding segment minimizes the number of red misclassifications. Recall the forbidden regions of a red line r are those regions in the parameter space in which corresponding segments misclassify r. We distinguish between five types of red lines, as in Figure 4:

- Line a intersects $\mathcal{U}(B^*)$ in points a_1 and a_2, with a_1 left of a_2. Only segments with ℓ_1^* left of a_1 or right of a_2 misclassify a. This produces two forbidden regions: $(-\infty, a_1) \times (-\infty, \infty)$ and $(a_2, \infty) \times (-\infty, \infty)$.
- Line b intersects $\mathcal{L}(B^*)$ in points b_1 and b_2, with b_1 left of b_2. Symmetric to line a this produces forbidden regions $(-\infty, \infty) \times (-\infty, b_1)$ and $(-\infty, \infty) \times (b_2, \infty)$.

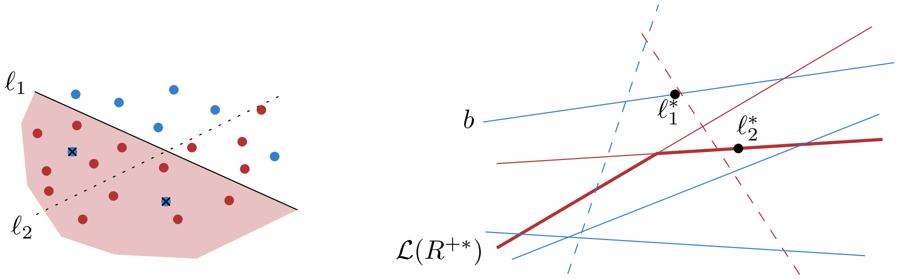


Figure 5 Left: in the primal we need to consider only the points above ℓ_1. Right: in the dual we need to consider only the lines below ℓ_1^*. In particular, ℓ_1^* should lie below (on) $\mathcal{L}(R^{+*})$.

- Line c intersects $\mathcal{U}(B^*)$ in c_1 and $\mathcal{L}(B^*)$ in c_2, with c_1 left of c_2. Only segments with endpoints after c_1 and before c_2 misclassify c. This produces the region $(c_1, \infty) \times (-\infty, c_2)$. (Segments with endpoints before c_1 and after c_2 do intersect c, but do not misclassify it.)
- Line d intersects $\mathcal{U}(B^*)$ in d_1 and $\mathcal{L}(B^*)$ in d_2, with d_1 right of d_2. Symmetric to line c it produces the forbidden region $(-\infty, d_1) \times (d_2, \infty)$.
- Line e intersects neither $\mathcal{U}(B^*)$ nor $\mathcal{L}(B^*)$. All segments misclassify e. In the primal this corresponds to red points inside the blue convex hull. This produces one forbidden region; the entire plane $\mathbb{R}^2$.

The forbidden regions generated by the red lines $r^* \in R^*$ divide the parameter space in axis-aligned orthogonal regions. Our goal is again to find a point with minimum *ply*, i.e. a point that is contained in the minimum number of these forbidden regions.

▶ **Lemma 5.1.** *Given a set $\mathcal{R}$ of n constant complexity, axis-aligned, orthogonal regions, we can compute the point with minimum ply in $O(n \log n)$ time.*

Proof sketch. We sweep through the plane with a vertical line z while maintaining a minimum ply point on z. See Figure 4 (right) for an illustration. We maintain the regions intersected by the sweep line in a slightly augmented segment tree [11]. In particular, each node v in the tree stores the size $s(v)$ of its canonical subset, the minimum ply $\text{ply}(v)$ within the subtree of v, and a point attaining this minimum ply. Since there are $O(n)$ forbidden regions (rectangles), each of which is added and removed once in $O(\log n)$ time, this leads to a running time of $O(n \log n)$. ◀

We construct $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ in $O(n \log n)$ time. For every red line r, we calculate its intersections with $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ in $O(\log n)$ time, determine its type $(a - e)$, and construct its forbidden regions. By Lemma 5.1 we can find a point with minimum ply in these forbidden regions in $O(n \log n)$ time. We thus obtain an $O(n \log n)$ time algorithm for finding an optimal East or West wedge. We can find an optimal North or South wedge in a similar manner, and thus obtain:

▶ **Theorem 5.2.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ containing all points of B and the fewest points of R in $O(n \log n)$ time.*

5.2 Wedge separation with blue outliers

We now consider the case where all red points must be classified correctly, and we minimize the number of blue outliers k_B. We show how to find an optimal North wedge; finding optimal South, East, or West wedges can be done analogously.

Fix line ℓ_1 and consider the problem of finding an optimal corresponding line ℓ_2. All points below ℓ_1 lie outside the North wedge, regardless of our choice of ℓ_2, and thus we have to consider only the points $B^+ \subseteq B$ and $R^+ \subseteq R$ above ℓ_1. See Figure 5. We do not allow red points in the North wedge, so ℓ_2 must lie above all red points R^+ and below as many blue points B^+ as possible. This is exactly the halfplane separation problem with blue outliers we solve in the full version in $O(n \log n)$ time. We can iterate through all $O(n^2)$ options for ℓ_1 (by walking through $\mathcal{A}(B^* \cup R^*)$) and compute the corresponding line ℓ_2 in $O(n \log n)$ time, which would lead to an $O(n^3 \log n)$ time algorithm. Below we describe an algorithm that avoids recomputing ℓ_2 from scratch every time, giving an $O(n^{5/2} \log n)$ time algorithm.

Let L be a set of lines, and let the *level* $l_L(p)$ of a point p (with respect to L) be the number of lines of L that lie below p. We define the level $l_L(s) = \max_{p \in s} l_L(p)$ of a segment s (with respect to L) as the maximum level of any point p on s.

Consider the dual, where we are looking for two points ℓ_1^* and ℓ_2^* such that no red line and as many blue lines as possible lie below both ℓ_1^* and ℓ_2^*. By Lemma 3.2 we can assume both ℓ_1^* and ℓ_2^* lie on a red-blue intersection. For a fixed point ℓ_1^* we are interested only in the set of lines B^{+*} and R^{+*} below ℓ_1^*, and since ℓ_2^* must lie below all of R^{+*} we can assume it lies on its lower envelope $\mathcal{L}(R^{+*})$. See Figure 5. The wedge North(ℓ_1, ℓ_2) correctly classifies exactly $l_{B^{+*}}(\ell_2^*)$ points. We are thus looking for the pair of points ℓ_1^* and ℓ_2^* that maximize the level $l_{B^{+*}}(\ell_2^*)$.

We now show that we can compute ℓ_2^* efficiently for every candidate point ℓ_1^*, provided that there is an oracle that can answer (a batch of) the following queries: given a point ℓ_1^* and a line segment s lying on a red line, compute the level $l_{B^{+*}}(s)$. We then show that we can implement an oracle that answers all $O(n^2)$ queries in $O(n^{5/2} \log n)$ time. This yields an $O(n^{5/2} \log n)$ time algorithm to compute an optimal north wedge.

Using an oracle to maintain ℓ_2^*. Consider any blue line $b \in B^*$ and assume w.l.o.g. that it is horizontal. We will shift ℓ_1^* from left to right along b, maintaining the set of red lines R^{+*} below ℓ_1^*. During this shift ℓ_1^* crosses each of the other lines at most once. We wish to maintain $\mathcal{L}(R^{+*})$ and a point with maximum level w.r.t. B^{+*} over all edges of $\mathcal{L}(R^{+*})$. Such a point corresponds to an optimal second point ℓ_2^* for the current point ℓ_1^*. By repeating this shift for every blue line $b \in B^*$ we consider all $O(n^2)$ candidate points for ℓ_1^* and their corresponding optimal point ℓ_2^*. This thus allows us to report an optimal solution.

We first show that we may keep an explicit representation of $\mathcal{L}(R^{+*})$ while shifting ℓ_1^* along b. We store the edges of $\mathcal{L}(R^{+*})$ in the leaves of a binary tree (ordered on increasing x-coordinate), which we refer to as the *explicit tree* of $\mathcal{L}(R^{+*})$. We then augment this explicit tree to additionally maintain the maximum level over all its edges. We show that we can maintain this tree in near-linear time in total; see the full version for details.

▶ **Lemma 5.3.** *While shifting ℓ_1^* along b we can maintain an explicit tree of $\mathcal{L}(R^{+*})$ in $O(n \log n)$ total time.*

With the explicit tree at hand, we require only $O(n)$ queries to the oracle during the entire shifting process to maintain an optimal point ℓ_2^*. Refer to the full version for details.

▶ **Lemma 5.4.** *We can maintain an edge of $\mathcal{L}(R^{+*})$ with maximum level w.r.t. B^{+*} while shifting ℓ_1^* along b using $O(n)$ queries to the oracle and $O(n \log n)$ additional time.*

Collecting queries to the oracle. What remains is to describe how to implement the oracle that answers the queries. Observe that the set of queries to the oracle is fixed. That is, the answer to an oracle query is independent of the answers to earlier queries, and the answers of queries do not influence which future queries will be performed. Therefore, we can perform a "dry"-run of the algorithm where we collect all queries, and then answer them in bulk. As we will see, this allows us to answer these queries efficiently.

Since each blue line generates $O(n)$ queries (Lemma 5.4), we have a total of $O(n^2)$ queries. Once we have the answers to all these queries, we once again run the algorithm for each blue line b. In this "real"-run of the algorithm we can answer queries in $O(1)$ time, and thus compute an optimal pair of points ℓ_1^* and ℓ_2^* with ℓ_1^* on b in $O(n \log n)$ time (Lemma 5.4). This then leads the following result.

▶ **Lemma 5.5.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge containing as many points of B as possible and no points of R in $O(T(n) + n^2 \log n)$ time, where $T(n)$ is the total time required to answer $O(n^2)$ oracle queries.*

Implementing the oracle. We will now show how to implement the oracle that can answer (a batch of) the following queries efficiently. Given a query (ℓ_1^*, s) consisting of a point ℓ_1^* and a red segment s, we wish to find the maximum level of any point on s w.r.t. the set B^{+*} of blue lines below ℓ_1^*. Maintaining the set B^{+*} and answering queries fully dynamically is difficult, so we will instead answer them in bulk.

We consider a red line r and assume w.l.o.g. that it is horizontal. Let Q be the set of queries whose line segment s lies on r, let $q_r = |Q|$, and let P be the set of query points ℓ_1^* corresponding to the queries in Q. See Figure 6.

We pick an arbitrary query $(\ell_1^*, s) \in Q$. Let $b \in B^{+*}$ be a blue line with negative slope; the other case is analogous. Consider the intersection point i between b and r. For points $p \in s$ left of i, b lies above p and thus b does not add to the level of p. For points $p \in s$ right of i, b lies below i and thus b does add to the level of p. Consider all intersections between r and lines in B^{+*}. We build a balanced binary tree on these intersections, ordered by x-coordinate, augmented such that each node also stores the point with the highest level in its subtree. Recall that s is a line segment on r, and thus represents an x-interval on r. We can easily answer the query (s, ℓ_1^*) by finding the $O(\log n)$ nodes representing that interval and returning the maximum level of any point inside their subtrees.

To answer the other queries we can shift the point ℓ_1^* through the arrangement $\mathcal{A}(B^*)$. When we cross into a different face, one blue line is inserted into or deleted from B^{+*}. We can update the binary tree in $O(\log n)$ time, meaning that when we reach a point $p \in P$ we can answer the corresponding query, again in $O(\log n)$ time. So, if we can walk through $\mathcal{A}(B^*)$ crossing s lines while visiting all points P, we can answer all queries Q in $O((s + |P|) \log n)$ time.

Consider a spanning tree on P. The stabbing number of a spanning tree is the maximum number of edges of the tree that can be intersected by a single line. With high probability (whp; in particular with probability $1 - 1/q_r^c$ for some arbitrarily large constant c) we can build a spanning tree T on P with stabbing number $O(\sqrt{q_r})$ in $O(q_r \log q_r)$ time [9]. Thus, each blue line intersects T at most $O(\sqrt{q_r})$ times, and therefore there are $O(n\sqrt{q_r})$ intersections between T and B (whp).

If we follow the spanning tree while walking through $\mathcal{A}(B^*)$ we will thus cross $O(n\sqrt{q_r})$ blue lines in total while visiting all points P, meaning we can answer all queries Q on r in $O(n\sqrt{q_r} \log n)$ time (note that the $O(q_r \log q_r)$ time to build the spanning tree is dominated by $O(n\sqrt{q_r} \log n)$ for any $q_r = O(n^2)$). Doing this for all lines $r \in R$ takes $O(\sum_r (n\sqrt{q_r} \log n))$ time. Recall we have $O(n^2)$ queries in total, so we have $\sum_r q_r = O(n^2)$. Since $\sqrt{\cdot}$ is a concave function, we have $\sqrt{a} + \sqrt{b} \leq \sqrt{2(a+b)}$ for any non-negative values a and

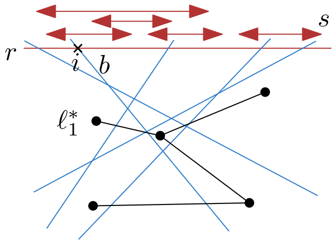


Figure 6 A set of queries on a red line r, with a spanning tree on P. Line b contributes to the level of all points right of i.

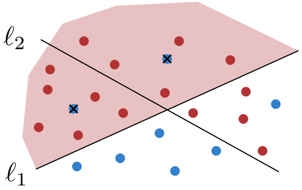


Figure 7 All points above ℓ_1 lie outside the South wedge. After fixing ℓ_1, we are left with a halfplane separation problem.

b. More generally, $\sum_i \sqrt{x_i} \le \sqrt{n\left(\sum_i x_i\right)}$ for any n non-negative values x_i. In particular, $\sum_r \sqrt{q_r} \le \sqrt{n\left(\sum_r q_r\right)} = O(n^{3/2})$. Therefore, we have an $O(\sum_r(n\sqrt{q_r}\log n)) = O(n^{5/2}\log n)$ time algorithm to answer all queries over all red lines.

▶ **Lemma 5.6.** *We can answer $O(n^2)$ queries in expected $O(n^{5/2}\log n)$ time.*

Together with Lemma 5.5 this yields an expected $O(n^{5/2}\log n + n^2\log n) = O(n^{5/2}\log n)$ time algorithm to find an optimal North wedge. We can symmetrically find an optimal South wedge by assuming the wedge lies below both ℓ_1 and ℓ_2. Similarly by assuming the wedge lies below ℓ_1 and above ℓ_2 we can find an optimal West or East wedge.

▶ **Theorem 5.7.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge containing as many points of B as possible and no points of R in expected $O(n^{5/2}\log n)$ time.*

5.3 Wedge separation with both outliers

We now consider the case where we allow and minimize both red and blue outliers. We show how to find an optimal South wedge; finding an optimal West, North, or East wedge can be done symmetrically. We first study the decision version of this problem: given an integer k', does there exist a South wedge $\mathcal{W}_B$ with at most k' outliers? We present an $O(nk'^2\log^3 n)$ time algorithm to solve this decision problem. Using exponential search to guess the optimal value k (i.e. guessing $k' = 1, 2, 4, 8\ldots$ and binary searching in the remaining interval) then leads to an $O(nk^2\log^3 n\log k)$ time algorithm to compute a wedge $\mathcal{W}_B$ that minimizes k.

In Lemma 5.8 we construct a small candidate set of lines that contains a line ℓ_1 that is used by an optimal wedge. Then our algorithm considers each line in this set and constructs an optimal wedge using it.

▶ **Lemma 5.8.** *In $O(nk'\log n)$ time, we can construct a set of $O(nk')$ lines that contains a line ℓ_1 used by an optimal wedge.*

Proof. Consider any line ℓ and suppose it is used in a South wedge as the line ℓ_1. Let B^+ and B^- be the set of blue points above, respectively below, ℓ_1. Since we are looking for a South wedge, all $k_1 = |B^+|$ blue points above ℓ_1 are misclassified, regardless of line ℓ_2 (see Figure 7). Therefore, any line with $k_1 > k'$ blue points above it is not a suitable candidate for ℓ_1. In the dual plane, this means ℓ_1^* must lie in the $(\le k')$*-level* $L_{\le k'}(B^*)$ *of* B^*, the set of points with at most k' lines of B^* below them. With a slight abuse of notation, we use $L_{\le k'}(B^*)$ to refer to the sub-arrangement of $\mathcal{A}(B^*)$ that lies in the $(\le k')$-level. The complexity of $L_{\le k'}(B^*)$ is $O(nk')$, and we can construct $L_{\le k'}(B^*)$ in $O(nk' + n\log n)$ time [12].

Consider any line $r \in R^*$, and observe that it intersects $L_{\leq k'}(B^*)$ at most $O(k')$ times. This follows from the fact that we can decompose $L_{\leq k'}(B^*)$ into $O(k')$ concave chains [8], and that r intersects each such chain at most twice. We can thus explicitly compute all the $O(nk')$ red-blue intersections in $L_{\leq k'}(B^*)$ in $O(nk' \log n)$ time, and by Lemma 3.2 these red-blue intersections contain the dual of a line used in an optimal wedge. ◀

Fix ℓ_1 to be any candidate line from Lemma 5.8. We wish to find another line ℓ_2 such that the wedge South(ℓ_1, ℓ_2) misclassifies at most k' blue points. Since all points above ℓ_1 are outside the wedge regardless of the line ℓ_2, we need to consider only the points B^- and R^- below ℓ_1. Recall that the choice of ℓ_1 already misclassifies k_1 blue points. Thus, we wish to find a line ℓ_2 that misclassifies at most $k_2 = k' - k_1$ points from B^- and R^-. That is, a line ℓ_2 such that the number of points from R^- below it plus the number of points from B^- above it is at most k_2. This is exactly the halfplane separation problem with both outliers, which we can solve using Chan's algorithm in $O((n + (k_2)^2) \log n)$ time [8]. Doing this for all $O(nk')$ candidate lines results in an $O(nk'(n + k^2) \log n) = O((n^2k' + nk'^3) \log n)$ time algorithm. Below we improve on this, by avoiding to recompute ℓ_2 from scratch every time.

Solving the halfplane separation problem dynamically. Consider again the set of candidate lines of Lemma 5.8, and in particular their dual points. By walking through the arrangement $L_{\leq k'}(B^*)$, we can visit all $O(nk')$ candidate points ℓ_1^* in $O(nk')$ steps, such that at each step we cross only one (red or blue) line. This means that only a single point is inserted in or deleted from the sets B^- and R^- per step. Rather than computing ℓ_2 from scratch after every step now, we maintain it dynamically.

We build the data structure of [13] that, given a value k', maintains a line ℓ_2 that misclassifies as few points from R^- and B^- as possible under insertions and deletions of red and blue points; if no line misclassifying at most k' points exists, the data structure reports this. The updates for the data structure have to be given in a 'semi-online' manner, which means that whenever a point is inserted we have to know when it is going to be deleted. This is not a problem in our case, since we can precompute all insertions and deletions on R^- and B^- by completing the walk through $L_{\leq k'}(B^*)$ before actually updating the data structure.

The data structure reports an optimal line ℓ_2 that misclassifies $k_2 \leq k'$ points of R^- and B^-, so the wedge South(ℓ_1, ℓ_2) then misclassifies $k_1 + k_2$ points. If $k_1 + k_2 \leq k'$ then we have found a wedge misclassifying at most k' points, so we are done with the decision problem, otherwise we move on to the next candidate for ℓ_1. If the data structure reports that there exists no line ℓ_2 misclassifying at most k' points, then we also move on to the next candidate.

The data structure has update time $O(k' \log^3 n)$ per insertion and deletion, and there are $O(nk')$ updates. Therefore, given a value k', we can find a South wedge with at most k' outliers, if it exists, in $O(nk'^2 \log^3 n)$ time. Using exponential search for the optimal value k then gives an optimal South wedge in $O(nk^2 \log^3 n \log k)$ time. As in the previous section, we can similarly find North, West, and East wedges.

▶ **Theorem 5.9.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ minimizing the total number of outliers k in $O(nk^2 \log^3 n \log k)$ time.*

With similar techniques to the above, we can improve (for some values of k_B) our algorithm for allowing blue outliers only to have an output-sensitive running time, see the full version for details.

▶ **Theorem 5.10.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ minimizing the number of blue outliers k_B in $O(nk_B^2 \log^2 n \log k_B)$ time.*

6 Separation with a double wedge

The final setting we study is that of finding a double wedge. We summarize our approach and results for all three cases.

Double wedge separation with red outliers. We consider finding a *bowtie* wedge $\mathcal{W}_B$ while minimizing red outliers, i.e. all of B and as little of R as possible lies in the West and East wedge. In the dual this corresponds to a line segment intersecting all of B^*, and as little of R^* as possible.

Observe that a segment intersecting all lines of B^* must have endpoints in antipodal outer faces of $\mathcal{A}(B^*)$, i.e. two opposite outer faces sharing the same two infinite bounding lines. For all $O(n)$ pairs of antipodal faces, we could apply a very similar algorithm to the wedge algorithm in Section 5.1, resulting in $O(n \cdot n \log n) = O(n^2 \log n)$ time.

Alternatively, we construct the entire arrangement $\mathcal{A}(B^* \cup R^*)$ of all lines explicitly in $O(n^2)$ time (see e.g. [11]). Consider a pair of faces P and Q that are antipodal in $\mathcal{A}(B^*)$, and assume w.l.o.g. they are separated by the x-axis, with P above Q. There are two types of red lines: *splitting* lines that intersect both P and Q once, and *stabbing* lines that intersect at most one of P and Q, see Figure 8. A red line is a splitting line for exactly one pair of antipodal faces, while it can be a stabbing line for multiple pairs. Recall that we wish to find a segment from P to Q intersecting as few red lines as possible. The s splitting lines divide the boundary of P and Q into $s+1$ chains $P_0..P_s$ ($Q_0..Q_s$). Within one such chain P_i on P we only need to consider the point p_i with the most stabbing lines above it: a segment from p_i to Q will not intersect those lines, since Q is below P_i. Similarly, we only need to consider point q_j on chain Q_j with the most stabbing lines below it. Using dynamic programming we can then find the pair of chains P_i, Q_j such that $\overline{p_i q_j}$ intersects the fewest red lines in $O(n + s^2)$ time. Doing so for all pairs of antipodal faces yields a total running time of $O(n^2)$.

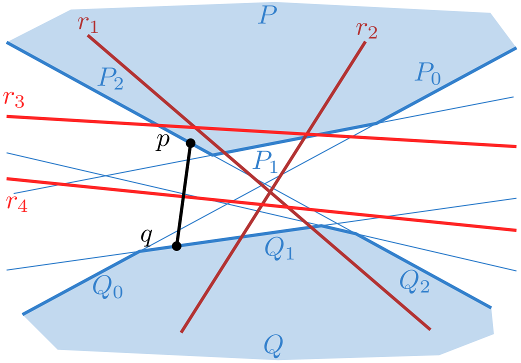


Figure 8 Two antipodal faces P and Q, with two splitting lines r_1, r_2 and two stabbing lines r_3, r_4, and an optimal segment $\overline{pq}$ from P to Q.

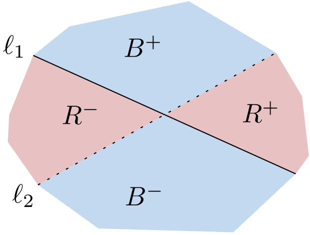


Figure 9 If we want B to lie in the North and South wedges, then B^+ and R^- should be above ℓ_2, and B^- and R^+ should be below ℓ_2.

▶ **Theorem 6.1.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct the bowtie double wedge $\mathcal{W}_B$ minimizing the number of red outliers k_R in $O(n^2)$ time.*

Double wedge separation with blue outliers. We wish to find a bowtie wedge $\mathcal{W}_B$ while minimizing blue outliers, i.e. none of R and as much of B as possible should lie in the West and East wedge. In the dual this corresponds to a line segment intersecting none of R^*, and as much of B^* as possible. This means the segment must lie in a single face of $\mathcal{A}(R^*)$. For each face F of $\mathcal{A}(R^*)$ we thus wish to find a segment intersecting as many blue lines as possible. Let B^*_F be the set of blue lines intersecting F. Then we can find such a segment in $O(|B^*_F| \log |B^*_F|)$ time with a parameter space approach similar to Section 5.1. We do this for each face F in $\mathcal{A}(R^*)$, yielding an $O(n^2 \log n)$ algorithm.

Double wedge separation with both outliers. We show how to find an hourglass wedge $\mathcal{W}_B$ while minimizing both types of outliers; by recolouring we can also find an optimal bowtie wedge. We use a similar approach as in Section 5.2, by considering a set of $O(n^2)$ candidate lines ℓ_1 and computing an optimal line ℓ_2 for each. For a fixed line ℓ_1, let B^+ and R^+ be the points above ℓ_1, and B^- and R^- be the points below ℓ_1, and let $P = B^+ \cup R^-$ and $Q = B^- \cup R^+$. See Figure 9. Then we wish to find a line ℓ_2 separating P and Q. Using the same dynamic datastructure for halfplane separation as used in Section 5.3, we can compute an optimal ℓ_2 for each ℓ_1 in $O(n^2 k \log^3 n \log k)$ time.

7 Concluding Remarks

We presented efficient algorithms for robust bichromatic classification of $R \cup B$ with at most two lines. Our results depend on the shape of the region containing (most of the) blue points B, and whether we wish to minimize the number of red outliers, blue outliers, or both. See Table 1. Several of our algorithms reduce to the problem of computing a point with minimum ply with respect to a set of regions. We can extend these algorithms to support weighted regions, and thus we may support classifying weighted points (minimizing the weight of the misclassified points). It is interesting to see if we can support other error measures as well.

There are also still many interesting open questions. Most prominently whether we can obtain faster algorithms for minimizing the number of blue outliers k_B or the total number of outliers k. Alternatively, it would be interesting to establish lower bounds for the various problems. In particular, are our algorithms for computing a halfplane minimizing k_R optimal, and in case of wedges (where the problem is asymmetric) is minimizing the number of blue outliers k_B really more difficult then minimizing k_R? For the strip case, the running time of our algorithm for minimizing k matches the worst case running time for halfplanes $(O((n + k^2) \log n)$, which is $O(n^2 \log n)$ when $k = O(n))$, but it would be interesting to see if we can also obtain algorithms sensitive to the number of outliers k.

References

1 Charu C. Aggarwal, editor. *Data Classification: Algorithms and Applications*. CRC Press, 2014. doi:10.1201/B17320.

2 Carlos Alegría, David Orden, Carlos Seara, and Jorge Urrutia. Separating bichromatic point sets in the plane by restricted orientation convex hulls. *Journal of Global Optimization*, 85(4):1003–1036, 2023. doi:10.1007/s10898-022-01238-9.

3 Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1&2):181–210, 1995. doi:10.1016/0304-3975(94)00254-G.

4 Esther M. Arkin, Delia Garijo, Alberto Márquez, Joseph S. B. Mitchell, and Carlos Seara. Separability of point sets by k-level linear classification trees. *International Journal of Computational Geometry & Applications*, 22(2):143–166, 2012. doi:10.1142/S0218195912500021.

5 Esther M. Arkin, Ferran Hurtado, Joseph S. B. Mitchell, Carlos Seara, and Steven Skiena. Some lower bounds on geometric separability problems. *International Journal of Computational Geometry & Applications*, 16(1):1–26, 2006. doi:10.1142/S0218195906001902.

6 Bogdan Armaselu and Ovidiu Daescu. Dynamic minimum bichromatic separating circle. *Theoretical Computer Science*, 774:133–142, 2019. doi:10.1016/j.tcs.2016.11.036.

7 Boris Aronov, Delia Garijo, Yurai Núñez Rodríguez, David Rappaport, Carlos Seara, and Jorge Urrutia. Minimizing the error of linear separators on linearly inseparable data. *Discrete Applied Mathematics*, 160(10-11):1441–1452, 2012. doi:10.1016/j.dam.2012.03.009.

8 Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM Journal on Computing*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.

9 Timothy M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012. doi:10.1007/s00454-012-9410-z.

10 Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in space of finite vc-dimension. *Discret. Comput. Geom.*, 4:467–489, 1989. doi:10.1007/BF02187743.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008.

12 Hazel Everett, Jean-Marc Robert, and Marc van Kreveld. An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 38–46, 1993. doi:10.1145/160985.160994.

13 Erwin Glazenburg, Frank Staals, and Marc van Kreveld. Robust classification of dynamic bichromatic point sets in r2, 2024. arXiv:2406.19161, doi:10.48550/arXiv.2406.19161.

14 Erwin Glazenburg, Thijs van der Horst, Tom Peters, Bettina Speckmann, and Frank Staals. Robust bichromatic classification using two lines, 2024. arXiv:2401.02897, doi:10.48550/arXiv.2401.02897.

15 Sariel Har-Peled and Vladlen Koltun. Separability with outliers. In *Proc. 16th International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2005. doi:10.1007/11602613_5.

16 Ferran Hurtado, Mercè Mora, Pedro A. Ramos, and Carlos Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1-2):110–122, 2004. doi:10.1016/j.dam.2003.11.014.

17 Ferran Hurtado, Marc Noy, Pedro A. Ramos, and Carlos Seara. Separating objects in the plane by wedges and strips. *Discrete Applied Mathematics*, 109(1-2):109–138, 2001. doi:10.1016/S0166-218X(00)00230-4.

18 Ferran Hurtado, Carlos Seara, and Saurabh Sethia. Red-blue separability problems in 3D. *International Journal of Computational Geometry & Applications*, 15(2):167–192, 2005. doi:10.1142/S0218195905001646.

19 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

20 D. Sculley and Gabriel M. Wachman. Relaxed online SVMs for spam filtering. In *Proc. 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 415–422. Association for Computing Machinery, 2007. doi:10.1145/1277741.1277813.

21 Carlos Seara. *On geometric separability.* PhD thesis, Univ. Politecnica de Catalunya, 2002.

22 Aihua Shen, Rencheng Tong, and Yaochen Deng. Application of classification models on credit card fraud detection. In *Proc. 2007 International conference on service systems and service management*, pages 1–4. IEEE, 2007.

Robust Classification of Dynamic Bichromatic Point Sets in $\mathbb{R}^2$

Erwin Glazenburg
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Let $R \cup B$ be a set of n points in $\mathbb{R}^2$, and let $k \in 1..n$. Our goal is to compute a line that "best" separates the "red" points R from the "blue" points B with at most k outliers. We present an efficient semi-online dynamic data structure that can maintain whether such a separator exists ("semi-online" meaning that when a point is inserted, we know when it will be deleted). Furthermore, we present efficient exact and approximation algorithms that compute a linear separator that is guaranteed to misclassify at most k, points and minimizes the distance to the farthest outlier. Our exact algorithm runs in $O(nk + n \log n)$ time, and our $(1+\varepsilon)$-approximation algorithm runs in $O(\varepsilon^{-1/2}((n+k^2)\log n))$ time. Based on our $(1+\varepsilon)$-approximation algorithm we then also obtain a semi-online data structure to maintain such a separator efficiently.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases classification, duality, data structures, dynamic, linear programming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.34

Related Version *Full Version*: https://arxiv.org/abs/2406.19161 [13]

Funding *Erwin Glazenburg*: Supported by the Dutch Research Council (NWO) under project OCENW.M20.135.

1 Introduction

Classification is a well known and well studied problem: given a training set of n data items with known classes, decide which class to assign to a new query item. Support Vector Machines (SVMs) [8] are a popular method for binary classification in which there are just two classes: *red* and *blue*. An SVM maps the input data items to points in $\mathbb{R}^d$, and constructs a hyperplane s that separates the red points R from the blue points B "as well as possible". Intuitively, it tries to minimize the distance from s to the set $X(s, B \cup R) \subseteq R \cup B$ of points *misclassified* by s while maximizing the distance to the closest correctly classified points. A red point $r \in R$ is misclassified if it lies strictly inside the halfspace s^+ above (left of) s, whereas a blue point $b \in B$ is misclassified if it lies strictly inside the halfspace s^- below s. See Figure 1 for an illustration. An SVM is typically modeled as a convex quadratic programming problem with linear constraints. However, this cannot provide guarantees on the number of misclassifications nor on the running time.[1] In practice, solving such optimization problems is possible, but it is computationally expensive as it involves $n + d$ variables [17]. This problem is magnified as training a high-quality classifier typically requires computing many

[1] When we restrict the coefficients in the SVM formulation to be rational numbers with bounded bit complexity such a problem can be solved in polynomial time [14, 23, 18]. However, it is unclear if they can be extended to allow for arbitrary real valued costs.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 34; pp. 34:1–34:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

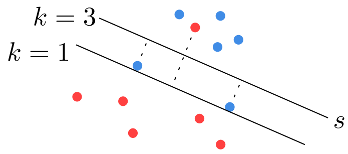


Figure 1 Red and blue points, and two optimal separators for $M_{\max}$ with 1 and 3 misclassification.

classifiers, each trained on a large subset of the input data, during cross-validation. Similarly, in streaming settings, the labeled input points arrive on the fly, and old data points should be removed due to concept drift [25]. Each such update requires recomputing the classifier. Hence, this limits the applicability of SVMs in these settings, even when the input data is low-dimensional.

The goal. We aim to tackle both these problems. That is, we wish to develop an "SVM-like" linear classifier that can provide guarantees on the number of misclassified points k, and can be constructed and updated efficiently. As the problem of minimizing k is NP-complete in general [1], we restrict our attention to the setting where the input points are low-dimensional. As it turns out, even for points in the plane, this is a challenging problem.

For a separator s, let $M_{\text{mis}}(s) = |X(s, B \cup R)|$ be the number of points misclassified by s. Let $S_k(B \cup R) = \{s \mid M_{\text{mis}}(s) \leq k\}$ denote the set of hyperplanes that misclassify at most k points from $B \cup R$, and let $dist(p, q)$ denote the Euclidean distance between geometric objects p and q. When the point sets are linearly separable, we want to compute a maximum-margin separator $s_{strip} \in S_0(R \cup B)$ that correctly classifies all points and maximizes the distance $M_{strip}(s_{strip}) = \min_{p \in R \cup B} dist(s_{strip}, p)$ to the closest points, exactly as in an SVM. Moreover, we would like to efficiently maintain such a separator when we insert or delete a point from $B \cup R$. The main challenge occurs when the point sets are not linearly separable. In this case, given a maximum k on the number of misclassified points, our aim is to find a separator $s_{\text{opt}} \in S_k(R \cup B)$ that minimizes the (Euclidean) distance $M_{\max}(s) = \max_{p \in X(s,B \cup R)} dist(p, s)$ to the furthest misclassified point. This thus asks for a minimum width strip containing the k outliers. We again would like to maintain such a separator when points are inserted or deleted. Furthermore, we may want to compute the smallest number $k_{\min}$ for which there exists a separator $s_{\min}$ that misclassifies at most $k_{\min}$ points. Note that decreasing the number of outliers may increase the value of $M_{\max}$, i.e. when $k_{\min} < k$ we may have $M_{\max}(s_{\min}) > M_{\max}(s_{\text{opt}})$, see Figure 1.

By the above discussion we distinguish four general variations of the problem:

MaxStrip: find a separator $s_{strip} = \text{argmax}_{s \in S_0(R \cup B)} M_{strip}(s)$
MinMax: find a separator $s_{\max} = \text{argmin}_s M_{\max}(s)$
MinMis: find a separator $s_{\text{mis}} = \text{argmin}_s M_{\text{mis}}(s)$
***k*-mis MinMax:** given a value k, find a separator $s_{\text{opt}} = \text{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$

Related Work. It is well known that for points in $\mathbb{R}^d$, for constant d, we can test if R and B can be linearly separated in $O(n)$ time by linear programming (LP) [22]. The problem becomes much more challenging when we allow a limited number of misclassifications. Everett et al. [12] show that for point sets R and B in the plane, one can find a line that separates R and B while allowing for at most k misclassifications in $O(n \log n + nk)$ time. Matoušek [20] shows how to solve LP-type problems while allowing at most k violated constraints. In particular, for linear programming in $\mathbb{R}^2$, his algorithm runs in $O(n \log n + k^3 \log^2 n)$ time. Chan [3] improves this to $O((n + k^2) \log n)$ time, and can compute the smallest number

k for which the points can be separated (the MinMis problem) in the same time. Aronov et al. [2] consider computing optimal separators with respect to other error measures as well. In particular, they consider minimizing the distance $M_{\max}(s)$ from s to the furthest misclassified point, as well as minimizing the average (squared) distance to a misclassified point $M_{\text{avg}}^{\beta}(s) = \sum_{p \in X(s,B \cup R)} (dist(s,p))^{\beta}$. For n points in $\mathbb{R}^2$, their running times for computing an optimal separator vary from $O(n \log n)$ for the $M_{\max}$ measure (the MinMax problem), to $O(n^{4/3})$ for the M_{avg}^1 measure, to $O(n^2)$ for M_{mis} (the MinMis problem) and the M_{avg}^2 measures. Some of their results extend to points in higher dimensions. Har-Peled and Koltun [16] consider similar measures, and present both exact and approximation algorithms. For example, they present an exact $O(nk^{d+1} \log n)$ time algorithm to find a hyperplane that minimizes the number of outliers (for points in $\mathbb{R}^d$), and an $O(n(\varepsilon^{-2} \log n)^{d+1})$ time algorithm to compute a $(1+\varepsilon)$-approximation of that number.[2] Their exact and approximation algorithms for computing a hyperplane minimizing $M_{\max}$ run in $O(n^d)$ and $O(n\varepsilon^{(d-1)/2})$ time, respectively. Matheny and Phillips [19] consider computing a separating hyperplane s, so that the discrepancy (the fraction of red points in s^- minus the fraction of blue points in s^-) is maximized. They present an $O(n + \varepsilon^{-d} \log^4(\varepsilon^{-1}))$ time algorithm that makes an additive error of at most ε (and thus misclassifies at most εn points more than an optimal (with respect to discrepancy) classifier).

Results. We can show that for points in $\mathbb{R}^1$ we can achieve both our goals: minimizing $M_{\max}$ with a hard guarantee on the number of outliers *and* efficiently supporting updates. In particular, in the full version [13] we present an optimal linear space solution:

▶ **Theorem 1.** *Let $B \cup R$ be a set of n points in $\mathbb{R}^1$. There is an $O(n)$ space data structure that, given a query value $k \in 1..n$ can compute an optimal separator $s_{\text{opt}} \in S_k(R \cup B)$ with respect to $M_{\max}$ in $O(\log n)$ time, and supports inserting or deleting a point in $O(\log n)$ time.*

The main focus of our paper is to establish whether we can achieve similar results for points in $\mathbb{R}^2$. If the points are separable, we can maintain a maximum-margin separator – essentially a maximum width strip – in $O(\log^2 n)$ time per update.

The problem gets significantly more complicated when the point sets are not separable, and we thus wish to compute, and maintain, a separator $s_{\text{opt}} \in S_k(B \cup R)$ minimizing the distance $M_{\max}$ to the farthest misclassified point. We can test whether a separator $s \in S_k(B \cup R)$ exists (and find the smallest k for which a separator exists) using LP with violations. In Section 3 we show how to dynamize Chan's approach [3] to maintain such a separator when the set of points changes. In particular, given a static linear objective function $f : \mathbb{R}^2 \to \mathbb{R}$ and a dynamic set H of halfplanes that is given in a semi-online manner (at the time we insert a halfplane h into H we are told when we will delete h), we show how to efficiently maintain a point p minimizing f that lies outside at most k halfplanes from H:

▶ **Theorem 2.** *Let H be a set of n halfplanes in $\mathbb{R}^2$, let f be a linear objective function, and let $k \in 1..n$. There is an $O(n + k^2 \log^2 n)$ space data structure that maintains a point p that violates at most k constraints of H (if it exists) and minimizes f, and supports semi-online updates in expected amortized $O(k \log^3 n)$ time.*

This then also allows us to maintain whether a separator that misclassifies at most k points exists in amortized $O(k \log^3 n)$ time per (semi-online) update, as well as maintain the minimum value k for which this is the case. Since linear programming queries have

[2] Here and throughout the rest of the paper, $\varepsilon > 0$ is an arbitrarily small constant.

many other applications, e.g. finding extremal points and tangents, we believe this result to be of independent interest. For example, given a threshold δ, our data structure also allows us to maintain a line ℓ that minimizes the number of points k at vertical distance exceeding δ from ℓ in amortized $O(k \log^3 n)$ time per update. Note that the best update time we can reasonably expect with this approach is $O((1 + k^2/n) \log n)$. For values of k that are small (e.g. polylogarithmic) or very large (near linear) our approach is relatively close to this bound.

In Section 4, we incorporate finding the best separator from $S_k(B \cup R)$; i.e. a separator that minimizes $M_{\max}$. We first tackle the algorithmic problem of computing such an optimal separator. Our main result here is:

▶ **Theorem 3.** *Let $B \cup R$ be a set of n points in $\mathbb{R}^2$, and let $k \in 1..n$. We can compute a separator $s_{\text{opt}} \in S_k(B \cup R)$ minimizing $M_{\max}$ in*

- *$O(nk + n \log n)$ time,*
- *$O((n + |S_k(B \cup R)| + k^3) \log^2 n)$ time, or*
- *when $k = k_{\min}$ in $O(k^{4/3} n^{2/3} \log n + (n + k^2) \log n)$ time.*

Here $|S_k(B \cup R)|$ denotes the complexity of (the region in the dual plane representing) $S_k(B \cup R)$. The key challenge is that this region $S_k(B \cup R)$ may consist of $\Theta(k^2)$ connected components, and each one has very little structure. While in the linear programming approach we can efficiently find one local minimum per connected component, that is no longer the case here. Instead, we explicitly construct the boundary of this region. Unfortunately, the total complexity of $S_k(B \cup R)$ is rather large: Chan [4] gives an upper bound of $|S_k(B \cup R)| = O(nk^{1/3} + n^{5/6-\varepsilon} k^{2/3+2\varepsilon} + k^2)$. We give two different algorithms to construct $S_k(B \cup R)$, and then efficiently find an optimal separator. When we restrict to the case where $k = k_{\min}$, i.e. finding an separator that minimizes $M_{\max}$ among all lines that misclassify the least possible number of outliers, each connected component of $S_k(B \cup R)$ is a single face in an arrangement of lines. This then gives us a slightly faster $O(k^{4/3} n^{2/3} \log^{2/3}(n/k) + (n + k^2) \log n)$ time algorithm as well.

Unfortunately, even when $k = k_{\min}$, dynamization appears very challenging. In the full version we present an $O((k^{4/3} n^{2/3} + n) \log^5 n)$ space data structure that supports insertions in amortized $O(kn^{3/4+\varepsilon})$ time, provided that the convex hulls of R and B remain the same. While the applicability of this result is limited, we use and develop an interesting combination of techniques here. For example, we develop a near linear space data structure that stores the the lower envelope of surfaces and allows for sub-linear time vertical ray shooting queries.

In Section 5, we slightly relax our goal and consider approximating the distance $M_{\max}$ instead. Our key idea is to replace the Euclidean distance by a convex distance function. This avoids some algebraic issues, as the distance between a point and a line now no longer has a quadratic dependency on the slope of the line. Instead the dependency becomes linear. We now obtain a much more efficient algorithm for finding a good separator:

▶ **Theorem 4.** *Let $B \cup R$ be a set of n points in $\mathbb{R}^2$, let $k \in 1..n$, and let $\varepsilon > 0$. We can compute a separator $s \in S_k(B \cup R)$ that is a $(1 + \varepsilon)$ approximation with respect to $M_{\max}$ in $O(\varepsilon^{-1/2}((n + k^2) \log n))$ time.*

We essentially "guess" the width δ of a strip "separating" the point sets, and show that we can use the linear programming machinery to efficiently test whether there exists such a strip containing at most k outliers . This involves extending the algorithm to deal with both "soft constraints" that may be violated, as well as "hard constraints" that cannot be violated, and using parametric search [21] to find the smallest δ for which such a strip exists.

▶ **Theorem 5.** *Let $B \cup R$ be a set of n points in $\mathbb{R}^2$, let $k \in 1..n$, and let $\varepsilon > 0$. There is an $O(\varepsilon^{-1/2}(k^2 \log^2 n + n))$ space data structure that maintains a separator $s \in S_k(B \cup R)$ that is a $(1+\varepsilon)$-approximation with respect to $M_{\max}$, and supports semi-online updates in expected amortized $O(\varepsilon^{-1/2} k \log^4 n)$ time.*

Applications. Our data structure from Theorem 5 can reduce the total time in a leave-out-one cross validation process by roughly a linear factor in comparison to Theorem 4. For m-fold cross validation we gain a factor m. Similarly, in a streaming setting in which we maintain a window of width w, we gain a factor of roughly w. Note that in both these settings, the semi-online updates indeed suffice.

2 Preliminaries

General definitions. We use the standard point-line duality that maps any point $p = (p_x, p_y)$ in the primal plane to a line $p^* : y = p_x x - p_y$ in the dual plane, and any line $\ell : y = mx + c$ in the primal plane into a point $(m, -c)$ in the dual plane.

Let A be a set of n lines, and let $k \in 1..n$. Let the *lower* $\leq$ *k-level* $L_{\leq k}(A) \subset \mathbb{R}^2$ of A be the set of points for which there are at most k lines below it. Similarly let the upper $\leq k$-level $L'_{\leq k}(A)$ be set of points for which there are at most k lines above it. Let $L_k(A)$ be the *k-level*, the boundary of $L_{\leq k}(A)$. Note that a k-level lies exactly on existing lines in A. Although these terms refer to a region in the plane, with a slight abuse of notation we will also use them to refer to the part of the arrangement $\mathcal{A}$ of the lines in A that lies in this region. The complexity of ($\mathcal{A}$ restricted to) $L_{\leq k}(A)$ is $O(nk)$, and it can be computed in $O(nk + n \log n)$ time [12]. Note that the lower 0-level $L_0(A)$ and the upper 0-level $L'_0(A)$ denote the *lower envelope* and the *upper envelope* of the set of lines, respectively.

In $O(n \log k)$ time we can compute a *concave chain decomposition* [3, 6] of $L_{\leq k}(A)$: a set of $O(k)$ concave chains of total complexity $O(n)$ that together cover all edges of $\mathcal{A}$ in $L_{\leq k}(A)$. See Figure 2a. A convex chain decomposition is defined similarly for $L'_{\leq k}(A)$.

Throughout this paper we assume points above a separating line s should be blue, and points below should be red. In the dual this means that lines above separating point s^* should be red, and lines below should be blue. In particular, we describe algorithms for finding the optimal separator that classifies in this way. We can then repeat the algorithm to find the best separator that classifies the other way around, and finally output the best of the two. For ease of description we assume all points in $R \cup B$ are in general position, meaning that all coordinates are unique, and no three points lie on a line.

Valid separators. Fix a value $k \in 1..n$. A separator s and its dual s^* are *valid* with respect to k if (and only if) $s \in S_k(B \cup R)$. Line s misclassifies all red points above s and all blue points below s. In the dual, this means all red lines below s^* and all blue lines above s^* are misclassified. Consider the dual arrangement of lines $R^* \cup B^*$. For any two separators s_1 and s_2 whose duals lie in the same face of the arrangement, $M_{\text{mis}}(s_1) = M_{\text{mis}}(s_2)$. Let a face containing valid points be a *valid face*, and note that points on the boundary of a valid face are also valid. A *valid region* is the union of a maximal set of adjacent valid faces, and the boundary of a valid region is composed of *valid edges* (note that we ignore edges that are fully contained within a valid region). Now observe that $S_k(B \cup R)$ thus corresponds to the union of these valid regions. With some abuse of notation we use $S_k(B \cup R)$ to refer to this union of regions in the dual plane as well.

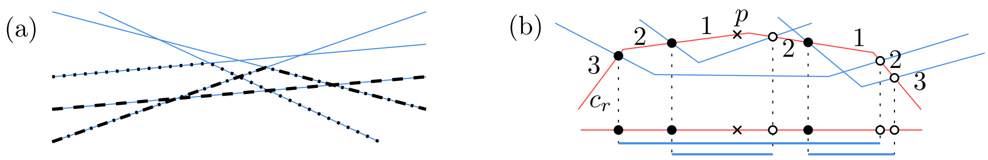


Figure 2 (a) A concave chain decomposition of $L_{\leq 2}(B^*)$. (b): Blue chains create intervals on a red chain c_r. The blue ply of a point p on c_r is the number of endpoints (open) before p plus the number of startpoints (closed) after p.

We observe the following useful properties (refer to the full version for omitted proofs):

▶ **Lemma 6** (Chan [4]). *The set $S_k(B \cup R)$ is contained in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, consists of $O(k^2)$ valid regions, and its total complexity is $O(nk^{1/3} + n^{5/6-\varepsilon}k^{2/3+2\varepsilon} + k^2)$.*

▶ **Lemma 7.** *There may be $\Omega(k^2)$ valid regions of total complexity $\Omega(k^2 + ne^{\sqrt{\log k}})$.*

▶ **Lemma 8.** *There are $O(k^2)$ red-blue intersections in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.*

▶ **Lemma 9.** *A line ℓ has $O(k)$ intersections with $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.*

▶ **Lemma 10.** *A valid region V is bounded by red lines on the top and blue lines on the bottom. The leftmost point of V is a red-blue intersection, or V is unbounded towards the left. The rightmost point in V is a red-blue intersection, or V is unbounded to the right.*

3 Dynamic linear programming with violations

In this section we consider the following problem: given a set of n constraints (halfplanes) H in $\mathbb{R}^2$, an objective function f, and an integer k, find a point p that violates at most k constraints and minimizes $f(p)$. We assume without loss of generality that $f(p) = p_x$, so we are looking for the leftmost *valid* point, that is, a point that violates at most k constraints. Chan solves this problem in $O((n+k^2)\log n)$ time [3]. In the same time bounds, their approach can find the minimum number $k_{\min}$ of constraints violated by any point. We give an overview of their techniques below, and then show how to make the approach semi-dynamic. We maintain an optimal point p under semi-online insertions and deletions of constraints. "Semi-online" means that when a constraint is inserted we are told when it will be deleted. We first do so for a given value k, and then extend the result to maintain $k_{\min}$.

The above linear programming problem is a generalization of (the dual of) our MinMis problem. Point p violates a constraint $h \in H$ if it lies outside of the halfplane. Let R be the set of lines bounding lower halfplanes, and B be the set lines bounding upper halfplanes. Point p violates all blue constraints above, and all red constraints below, and thus p violates exactly the $M_{\text{mis}}(p)$ lines in $X(p, R \cup B)$. This means we can solve the MinMis problem and compute $s_{\text{mis}} = \text{argmin}_s M_{\text{mis}}(s)$ in $O((n + k^2)\log n)$ time.

3.1 Chan's algorithm

Chan considers the decision version of the problem: given an integer k, find the leftmost point that violates at most k constraints. Their algorithm actually generates all local minima that violate fewer than k constraints as well, so by guessing $k = \sqrt{n}, 2\sqrt{n}, 4\sqrt{n}\ldots$ we can find the minimum value $k_{\min}$ for which a valid point exists in the same time bounds.

We first assume that there are no valid regions that are unbounded towards the left. By Lemma 10, the leftmost valid point in a valid region must then be a red-blue intersection, and by Lemma 8 there are only $O(k^2)$ of them. We construct the concave chain decomposition of $L_{\leq k}(R)$ and the convex chain decomposition of $L'_{\leq k}(B)$ in $O(n \log n)$ time, and compute their intersections in $O(k^2 \log n)$ time; this gives us all candidate optima.

Consider a red chain c_r, as in Figure 2b. Every blue chain c_b defines a (possibly empty) interval on c_r, such that points inside the interval lie above c_b and points outside the interval lie below c_b. The *blue ply* of a point p on c_r is the number of blue chains above p (and thus the number of violated blue constraints above p). This is the number of blue intervals not containing p, and thus the number of intervals ending before p or starting after p. By storing the start points and end points of all blue intervals in two balanced binary trees we can thus find the blue ply of any point p on c_r in $O(\log k)$ time. We call this the *chromatic ply data structure* of c_r. The chromatic ply data structure of a blue chain c_b is defined symmetrically.

For an intersection point p between red chain c_r and blue chain c_b we can now calculate its $M_{\text{mis}}(p)$ value: query c_r for the blue ply and query c_b for the red ply, both in $O(\log k)$ time, and sum them up. For all $O(k^2)$ red-blue intersections this takes $O(k^2 \log k)$ time. Among them we then find the leftmost valid intersection and return it, if it exists.

If the optimum was unbounded, then part of the leftmost segment of one of the chains must be valid. We can check this in $O(k \log k)$ time using the chromatic ply data structures.

3.2 A semi-dynamic data structure for a fixed k

We now make the above algorithm dynamic under semi-online insertions and deletions: given a fixed value k, we maintain the leftmost point that violates at most k constraints.

We first show how to maintain the concave chain decomposition of $L_{\leq k}(R)$ (and similarly the convex chain decomposition of $L'_{\leq k}(B)$) using an extension of the logarithmic method [10, 26], then show how to maintain the chromatic ply datastructures, and lastly use these chains to actually maintain the leftmost valid separator.

Maintaining the concave chain decomposition. We maintain the concave chain decomposition of $L_{\leq k}(R)$ using Dobkin and Suri's extension of the logarithmic method [10, 26]. We maintain a partition of R into $z = O(\log n)$ subsets $R_0, R_1..R_z$, such that for each layer i:

(1) none of the lines in set R_i will be deleted for at least 2^i updates after the set is created.

(2) $|R_i| = O(2^i)$.

For each set R_i we store the concave chain decomposition of $L_{\leq k}(R_i)$. Since each such structure contains $O(k)$ chains, we have $O(k \log n)$ chains in total. The union of these chains also covers $L_{\leq k}(R)$: if a line ℓ is among the lowest k lines in R at some x-coordinate, it must also be among the lowest k lines in any subset $R' \subseteq R$, including the subset R_i containing ℓ.

The basic idea is the following. After set R_i is created, by condition (1) no items will be deleted from it for at least 2^i updates, so it remains fixed for 2^i updates and gets rebuilt after that. As such, the smaller data structures are rebuilt quite often, and the larger data structures remain fixed for a long time. By construction, deletions happen only at layer 0 from set R_0, which contains $O(1)$ lines. Lines are inserted in layer 0, and gradually move to higher layers where they remain fixed for an ever increasing number of updates. When a line is to be deleted soon, it gradually moves down to layer 0 again.

▶ **Lemma 11.** *We can maintain $O(k \log n)$ concave chains of total complexity $O(n)$ that cover $L_{\leq k}(R)$ under semi-online insertions and deletions in $O(\log^2 n)$ amortized time.*

In fact, we can maintain a slightly altered version of the above data structure within the same time and space bounds. Let 2^x be the smallest power of two that is at least $k \log n$, i.e. $2^{x-1} < k \log n \leq 2^x$. We store the 2^x lines that are the first to be deleted in a separate list, the *leftover* list. We do not build a chain data structure on the leftover lines, but instead we let each leftover line forms a trivial chain, so we still have $O(k \log n)$ chains covering $L_{\leq k}(R)$. This way all sets R_j with $j < x$ are empty. We can thus perform 2^x *cheap* updates without having to modify any of the sets R_j: we can simply insert directly in (or delete directly from) the leftover list, without having to rebuild any data structure, in $O(1)$ time. Once every 2^x updates we perform an *expensive* update, destroying all sets up to some layer i' (the largest set that no longer adheres to invariant (1)), and redistributing all the lines in them over the layers 0 through i' again. Each set R_i still has an amortized update time of $O(i)$, and thus the total amortized update time remains $O(\log^2 n)$.

Maintaining intersections and chromatic ply data structures. Next, we show how to maintain the chromatic ply data structures on the chains, which also gives us the set $I_{\leq k}$ of $O(k^2)$ red-blue intersections in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.

We maintain a concave chain decomposition of $L_{\leq k}(R^*)$ and a convex chain decomposition of $L'_{\leq k}(B^*)$ using Lemma 11, with $O(k \log n)$ leftover lines. Whenever we perform an expensive update on one of the two (i.e. rebuilding the chains), we do so on the other as well. On each red chain c_r we maintain a blue chromatic ply data structure. Similarly on each blue chain c_b we maintain a red ply data structure. Additionally, we maintain a set I of the $O(k^2 \log^2 n)$ red-blue intersections between the chains. This is a superset of $I_{\leq k}$.

Consider the insertion of a line r. Depending on the type of update, we do the following:

Cheap update: line r is added to the leftover list, and forms a trivial chain c_r. We compute all $O(k \log n)$ intersections between c_r and the blue chains, insert them in I, and build the blue ply data structure on c_r. For each intersected blue chain c_b we insert the interval induced on c_b by c_r into the red ply data structure of c_b. This takes $O(k \log^2 n)$ time.

Expensive update: some number of chains are rebuilt. We rebuild the set I and the chromatic ply data structures on all chains from scratch. Since we have $O(k \log n)$ red and blue chains, this takes $O(k^2 \log^3 n)$ time.

Every $2^x = O(k \log n)$ updates we have $2^x - 1$ cheap updates, taking $O(k \log^2 n)$ time each, and one expensive update, taking $O(k^2 \log^3 n)$ time. This thus takes $O(k^2 \log^3 n)$ time for 2^x updates, making the amortized updates time $O(k \log^2 n)$. We thus have:

▶ **Lemma 12.** *We can maintain a set $I \supseteq I_{\leq k}$ of $O(k^2 \log^2 n)$ bichromatic intersection points under semi-online updates in amortized $O(k \log^2 n)$ time. This uses $O(n + k^2 \log^2 n)$ space.*

Maintaining the leftmost valid point. The last step is to maintain the leftmost valid point s for a fixed value k. We know s is contained in the set I maintained by Lemma 12, but simply iterating through the entire set each update would take too long. We store I in a data structure that maintains $M_{\text{mis}}(p)$ for each $p \in I$, and can handle the following operations:

Insertion/Deletion: Inserting or deleting a point (a red-blue intersection).

Halfplane update: Update $M_{\text{mis}}(p)$ for each $p \in I$ after the insertion or deletion of a constraint, e.g. increment $M_{\text{mis}}(p)$ by one for all points p in the halfplane above an inserted line r (or in the halfplane below an inserted line b).

Query: Given a query value $k' \leq k$, return the leftmost point $p \in I$ with $M_{\text{mis}}(p) \leq k'$.

We can achieve the above using a binary search tree on I sorted by x-coordinate, and a partition tree [5] on I where each node u stores (a point attaining) the minimum number of constraints violated by a point in its canonical subset. We use the logarithmic method to handle insertions on the partition tree, and perform deletion of a point $p \in I$ implicitly by setting $M_{\text{mis}}(p) = \infty$. A halfplane update corresponds to one "query" in the partition tree, where we only recurse on intersected triangles. Using the binary search tree and the partition tree, we can then binary search for the leftmost point that violates at most k' constraints.

▶ **Lemma 13.** *We can build a data structure on a set of $O(k^2 \log^2 n)$ points I that can perform insertions and deletions in $O(\log k \log n)$ amortized time, halfplane updates in expected $O(k \log^2 n)$ time, and queries in expected $O(k \log^3 n)$ time. The data structure uses $O(k^2 \log^2 n)$ space.*

We can now dynamically maintain the solution to an LP with at most k violations by using Lemmas 11, 12 and 13 as follows. For each cheap update, e.g. the insertion of a line r, we find the $O(k \log n)$ intersections between r and blue chains, and insert them in $O(k \log^2 n \log k)$ total time. We then perform one halfplane update in $O(k \log^2 n)$ time. For each expensive update we discard the data structures from Lemmas 12 and 13 and rebuild them from scratch. This takes $O(k^2 \log^3 n)$ time, and thus $O(k \log^2 n)$ amortized time. After every update we perform one query with $k' = k$ in $O(k \log^3 n)$ time. This establishes Theorem 2. In the full version, we extend the data structure to maintain the minimum number k_{min} of constraints violated by any point as well.

4 Exact algorithms for k-mis MinMax

For the k-mis MinMax problem we are given point sets R and B and an integer k and wish to compute a separator $s_{\text{opt}} = \operatorname{argmin}_{s \in S_k(R \cup B)} M_{\max}(s)$ that misclassifies at most k points and minimizes the distance to the furthest misclassified point. In this section we present an exact algorithm for this problem. In Section 4.1 we first discuss some useful geometric properties. In Section 4.2 we then present algorithms to construct the valid regions $S_k(R \cup B)$. Finally, in Section 4.3, we show how we can then compute an optimal separator.

4.1 Geometric properties

The MinMax problem. We first consider the MinMax problem. Here, we wish to compute $s_{\max} = \operatorname{argmin}_s M_{\max}(s)$, a separator with minimal distance to the farthest misclassified point. Consider the dual plane. At a fixed x-coordinate, this point lies exactly in the middle of the envelopes $L_0(R^*)$ and $L'_0(B^*)$. Let the MinMax curve be the polygonal curve in the middle of $L_0(R^*)$ and $L'_0(B^*)$, and observe that it consists of $O(n)$ segments. See Figure 3.

▶ **Lemma 14.** *Let s be a point in the interior of an edge e of the MinMax curve. Moving s left or moving s right along e decreases the error $M_{\max}(s)$.*

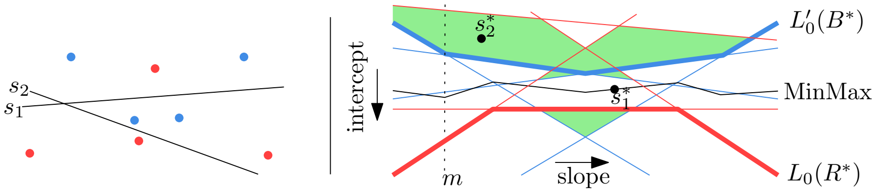


Figure 3 Some primal points (left) with their dual (right). Valid faces for $k = 2$ are green.

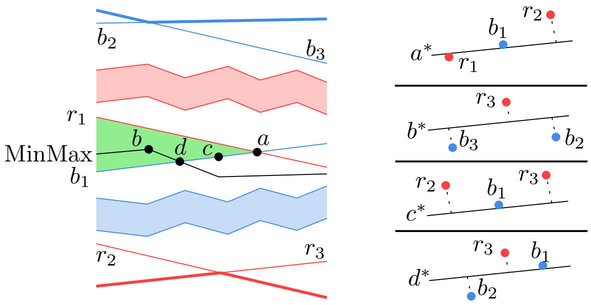


Figure 4 Left: the cases **a**, **b**, **c**, **d** for s^*_{opt} in the dual plane; the red/blue regions represent some number of correctly classified lines. Right: the cases **a**, **b**, **c**, **d** for s_{opt} in the primal plane.

The k-min MinMax problem. For the k-mis MinMax problem, we wish to compute $s_{\text{opt}} = \text{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$, a valid separator with minimal $M_{\max}(s_{\text{opt}})$. At a fixed x-coordinate, this is the valid separator (if it exists) with the smallest vertical distance to the MinMax curve. This fact and Lemma 14 lead to the following characterization:

▶ **Lemma 15.** *A point s^*_{opt} dual to an optimal separator is one of the following:*
a. *A vertex of a valid face, vertically closest to MinMax.*
b. *A (valid) vertex of MinMax.*
c. *The first valid point directly above or below a vertex of MinMax.*
d. *The intersection of a MinMax edge e with a valid edge, closest to one of e's endpoints.*

Proof sketch. In the primal plane, the optimal separator s_{opt} has to be "bounded" by at least three points, otherwise we can rotate or translate s_{opt} slightly to decrease $M_{\max}(s_{\text{opt}})$. These bounding points can either be extremal points that we want the separator to be as close to as possible, or points that the separator is not allowed to cross because that would make it invalid. The four ways in which s_{opt} can be bounded are shown in Figure 4. ◀

4.2 Constructing the valid regions

We present three algorithms for constructing the valid regions $S_k(B \cup R)$.

First, by Lemma 6 all valid points lie inside $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, so we present a simple algorithm that constructs this part of the arrangement, and prunes all invalid regions. Since $L_{\leq k}(R^*)$ and $L'_{\leq k}(B^*)$ have complexity $O(nk)$, this gives an $O(n \log n + nk)$ time algorithm.

Second, we present a much more involved output sensitive $O((nk^{1/3} + n^{5/6-\varepsilon}k^{2/3+2\varepsilon} + k^3)\log^2 n)$ time algorithm, which is faster for $k < n^{2/3}$. It is based on an approach sketched by Chan [4] to compute the valid region in an output-sensitive manner, by first computing the bichromatic intersection points of $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, and then tracing $S_k(B \cup R)$, starting from these bichromatic intersection points "as in a standard k-level algorithm". They claim this results in a running time of $O(|S_k(B \cup R)| \operatorname{polylog} n)$ time, but do not provide details. The k-level in an arrangement of lines is connected, whereas here $S_k(B \cup R)$ may consist of $\Omega(k^2)$ disconnected pieces (Lemma 7). This unfortunately provides some additional difficulties in initializing the data structure used in the tracing process. Hence, it is not clear that it can indeed be done in $O(|S_k(B \cup R)| \operatorname{polylog} n)$ time. Instead, we present an algorithm that runs in $O((|S_k(B \cup R)| + n + k^3)\log^2 n)$ time, the stated time bound.

Finally, if we care only about the case where $k = k_{min} = M_{\text{mis}}(s_{\text{mis}})$, i.e. where we are required to misclassify as few points as possible, we observe that each valid region is a single face. By Lemma 6 there are $O(k^2)$ valid regions, so now there are $O(k^2)$ valid faces. Clarkson et al. [7] show that m faces in an arrangement have a complexity of $O(m^{2/3}n^{2/3} + n)$, and we can construct them in $O(k^{4/3}n^{2/3}\log^{2/3}(n/k) + (n + k^2)\log n)$ time [27].

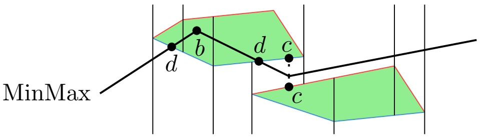


Figure 5 Two valid faces with their vertical decomposition and type b, c and d points.

4.3 An algorithm for solving the k-mis MinMax problem

We now show how, given the valid regions, we can compute an optimal separator $s_{\text{opt}} \in S_k(B \cup R)$ efficiently. We start by constructing $L_0(R)$ and $L'_0(B)$, and simultaneously scan through them to construct the MinMax curve $s^*_{\max}$. This takes $O(n \log n)$ time [9]. By Lemma 15 an optimal separator is of type a, b, c, or d. So, we will now compute all these candidate optima, and iterate through them to find the one with lowest error.

Type a points. Since we are given $S_k(B \cup R)$, we can simply scan through its vertices, keeping track of the vertex with the smallest error. To calculate the error of a vertex, we need to know which segment of $s^*_{\max}$ it lies above/below; then the error can be calculated in $O(1)$ time. We can compute this in $O(\log n)$ time per vertex using binary search (since MinMax is x-monotone). Hence, this step takes $O(|S_k(B \cup R)| \log n)$ time.

Type b and c points. Recall type b points are MinMax vertices, and type c points are the first valid points above or below MinMax vertices. We construct the *trapezoidal decomposition* of $S_k(B \cup R)$ in $O(|S_k(B \cup R)| \log n)$ time, which supports $O(\log n)$ time point location queries [24]. Each trapezoid has vertical left and right sides, see Figure 5.

For each vertex of MinMax we perform one point location query, which tells us what trapezoid the vertex lies in. If this trapezoid is inside a valid region, the vertex is a type b point. Otherwise the closest valid edges vertically above and below this vertex are simply the edges bounding that trapezoid, giving us up to two type c points. Since MinMax has $O(n)$ vertices, this gives us all type b and c points in $O(n \log n)$ time. Including the time to build the decomposition, this thus takes $O((|S_k(B \cup R)| + n) \log n)$ time.

Type d points. Recall type d points are intersections between MinMax and edges bounding $S_k(B \cup R)$. In particular, for every MinMax edge we care only about its outermost intersection points. We walk along MinMax from left to right through the vertical decomposition until we find the leftmost intersection on an edge; we then continue the walk from the next MinMax vertex. We find the rightmost intersection symmetrically. After locating the MinMax vertices in $O(n \log n)$ time, this takes $O(n + |S_k(B \cup R)|)$ time, since MinMax is x-monotone.

▶ **Lemma 16.** *Given $S_k(B \cup R)$, we can compute a separator $s_{\text{opt}} = \operatorname{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$ in $O((|S_k(B \cup R)| + n) \log n)$ time.*

By combining Lemma 16 with the three algorithms from Section 4.2 we thus obtain an $O((nk + n) \log n)$ (which we can reduce to $O(nk + n \log n)$) and an $O((|S_k(B \cup R)| + n + k^3) \log^2 n)$ time algorithm for the general problem, and an $O(k^{4/3} n^{2/3} \log n + (n + k^2) \log n)$ time algorithm for when $k = k_{\min}$. These results together establish Theorem 3.

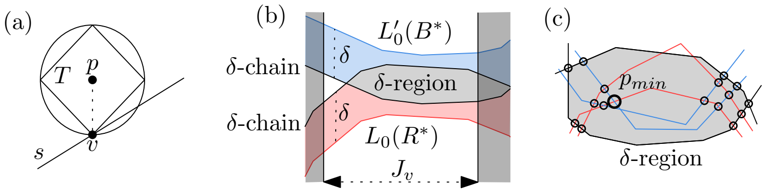


Figure 6 (a) The Euclidean unit circle and convex unit 4-gon. (b) The convex and concave δ-chain forming a δ-region. (c) A δ-region, with candidate red-blue intersections marked.

5 An ε-approximation algorithm

Let $s_{\text{opt}} \in S_k(B \cup R)$ be an optimal valid separator minimizing $M_{\max}$, and let $\varepsilon \in (0, 1)$ be some given threshold. Our goal is to compute a $(1+\varepsilon)$-*approximation* of s_{opt}: that is, we want to find a valid separator $\hat{s}$ with $M_{\max}(\hat{s}) \leq (1+\varepsilon)M_{\max}(s_{\text{opt}})$. The main idea is to replace the Euclidean distance function *dist* by some convex distance function $\hat{d}$ that approximates *dist*, and compute a separator $\hat{s}$ that minimizes $\hat{M}(\hat{s}) = \max_{p \in X(\hat{s}, B \cup R)} \hat{d}(p, \hat{s})$.

Let p be a point and s be a line, let $t = \Theta(1/\sqrt{\varepsilon})$, and let T be a convex regular t-gon centered at the origin inscribed by a unit disk. See Figure 6a. We then define the convex distance function $\hat{d}(p, s) = \min\{\lambda \mid s \cap (p + \lambda T) \neq \emptyset\}$ to be the smallest scaling factor for which a scaled copy of T centered at p intersects s. It can be shown that $dist(p, s) \leq \hat{d}(p, s) \leq (1+\varepsilon)dist(p, s)$ [11, 15], and thus $M_{\max}(s) \leq \hat{M}(s) \leq (1+\varepsilon)M_{\max}(s)$. It follows that the separator $\hat{s}$ minimizing $\hat{M}$ is a $(1+\varepsilon)$-approximation of s_{opt}.

Observe that this distance $\hat{d}(p, s)$ is realized in a corner v of the t-gon; i.e. the t-gon scaled by a factor $\hat{d}(p, s)$ intersects s in a corner point v of the t-gon. We say v is a *realizer* for the line s. More specifically, there is some interval of slopes J_v such that v is the realizer for all lines with a slope in the interval J_v. For each slope interval J_v we will compute a valid separator $\hat{s}^v$ with slope in J_v minimizing $\hat{M}(\hat{s}^v)$, and finally $\hat{s} = \text{argmin}_v M_{\max}(\hat{s}^v)$.

We consider one such slope interval J_v. Assume w.l.o.g. that v is vertically below the center point of the t-gon (we can rotate the plane to achieve this). This means interval J_v is centered at slope 0, so $J_v = (-\pi/t, \pi/t)$, and the distance $\hat{d}(p, s)$ between a point p and line s is the vertical distance between p and s. Since vertical distance is preserved by dualizing, this means that for all points s in the x-interval J_v, the value $\hat{M}(s)$ expresses the vertical – and thus convex t-gon – distance from s to $L_0(R^*)$ or $L'_0(B^*)$, whichever is larger.

The algorithmic problem. Extending Chan's algorithm from Section 3.1, we build a data structure that, for a given value δ, can find a valid separator $s \in J_v \times \mathbb{R}$ with $\hat{M}(s) \leq \delta$ if it exists. We then use parametric search [21] to find the optimal value δ, and a separator $\hat{s}^v$.

Fix a value δ, and observe that all points with error at most δ lie at most δ below $L'_0(B)$. This can be imagined as moving $L'_0(B)$ down by δ. Let the resulting chain be the *convex δ-chain*, see Figure 6(b). Similarly, let the *concave δ-chain* be $L_0(R)$ moved up by δ. All points with error at most δ must thus lie above the convex δ-chain, and below the concave δ-chain: the *δ-region*. The question now becomes: does a valid point exist in the δ-region?

In Section 3.1 we considered only red-blue intersections. Similarly, we can show that now we need to consider only intersections between convex chains (a blue chain or the convex δ-chain) and concave chains (a red chain or the concave δ-chain). There are $O(k^2)$ such convex-concave intersections (see Figure 6(c)). We can compute them all during preprocessing, find the valid point $p_{\min}$ among them with smallest error, and simply forget about all others.

The data structure consists of three parts. First, a concave chain decomposition of $L_{\leq k}(R)$, and a convex chain decomposition of $L'_{\leq k}(B)$, with a chromatic ply data structure for every chain. Second, the point $p_{\min}$. Third, the envelopes $L_0(R)$ and $L'_0(B)$. This can all be built in $O((n+k^2)\log n)$ time using Chan's method, and uses $O(n+k^2)$ space.

We answer a query with value δ as follows. We check if $\hat{M}(p_{\min}) \leq \delta$, and if so, return $p_{\min}$. If not, we find the $O(k)$ convex-concave intersections involving the δ-chains, and build a red ply data structure for the convex δ-chain, and a blue ply data structure for the concave δ-chain. For each intersection p we compute $M_{\text{mis}}(p)$ using the chromatic ply data structures, and compute $M_{\max}(p)$ using the envelopes. Finally we return the valid intersection with lowest error if its error is at most δ, otherwise there exists no point with error at most δ.

Using parametric search on the above data structure gives us a valid separator with slope in J_v with the lowest error in $O((n+k^2)\log n)$ time. Doing this for all $t = \Theta(1/\sqrt{\varepsilon})$ slope intervals J_v proves Theorem 4.

A dynamic data structure. Using the dynamic chain decomposition data structure from Lemma 11, we can maintain the data structure under semi-online updates, and perform the parametric search after every update to maintain an optimum $\hat{s}$, thus proving Theorem 5.

References

1 Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theor. Comput. Sci.*, 147(1&2):181–210, 1995. doi:10.1016/0304-3975(94)00254-G.

2 Boris Aronov, Delia Garijo, Yurai Núñez Rodríguez, David Rappaport, Carlos Seara, and Jorge Urrutia. Minimizing the error of linear separators on linearly inseparable data. *Discret. Appl. Math.*, 160(10-11):1441–1452, 2012. doi:10.1016/j.dam.2012.03.009.

3 Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.

4 Timothy M. Chan. On the bichromatic k-set problem. *ACM Trans. Algorithms*, 6(4):62:1–62:20, 2010. doi:10.1145/1824777.1824782.

5 Timothy M Chan. Optimal partition trees. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 1–10, 2010. doi:10.1145/1810959.1810961.

6 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discret. Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/S00454-016-9784-4.

7 Kenneth L Clarkson, Herbert Edelsbrunner, Leonidas J Guibas, Micha Sharir, and Emo Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete & Computational Geometry*, 5(2):99–160, 1990. doi:10.1007/BF02187783.

8 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995. doi:10.1007/BF00994018.

9 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

10 David Dobkin and Subhash Suri. Dynamically computing the maxima of decomposable functions, with applications. In *30th Annual Symposium on Foundations of Computer Science*, pages 488–493. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63523.

11 Richard M Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.

12 Hazel Everett, Jean-Marc Robert, and Marc J. van Kreveld. An optimal algorithm for the ($\leq$ k)-levels, with applications to separation and transversal problems. *Int. J. Comput. Geom. Appl.*, 6(3):247–261, 1996.

13 Erwin Glazenburg, Frank Staals, and Marc van Kreveld. Robust classification of dynamic bichromatic point sets in r2, 2024. arXiv:2406.19161, doi:10.48550/arXiv.2406.19161.

14 D. Goldfarb and S. Liu. An $O(n^3L)$ primal interior point algorithm for convex quadratic programming. *Mathematical Programming*, 49(1):325–340, 1990.

15 Sariel Har-Peled and Mitchell Jones. Proof of dudley's convex approximation. *arXiv preprint*, 2019. arXiv:1912.01977.

16 Sariel Har-Peled and Vladlen Koltun. Separability with outliers. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005, Proceedings*, volume 3827 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2005. doi:10.1007/11602613_5.

17 Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In *International conference on machine learning*, pages 566–574. PMLR, 2014. URL: http://proceedings.mlr.press/v32/hsieha14.html.

18 M. K. Kozlov, S. P. Tarasov, and L. G. Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Comp. Math. and Math. Phys.*, 20(5):223–228, 1980.

19 Michael Matheny and Jeff M. Phillips. Approximate maximum halfspace discrepancy. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.4.

20 Jirí Matousek. On geometric optimization with few violated constraints. *Discret. Comput. Geom.*, 14(4):365–384, 1995. doi:10.1007/BF02570713.

21 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

22 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

23 R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part II: Convex quadratic programming. *Mathematical Programming*, 44(1):43–66, 1989. doi:10.1007/BF01587076.

24 Neil Sarnak and Robert E Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.

25 Jeffrey C Schlimmer and Richard H Granger Jr. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, pages 502–507, 1986.

26 Michiel Smid. A worst-case algorithm for semi-online updates on decomposable problems. Technical report, Univeristät des Saarlandes, 1990. doi:10.22028/D291-26450.

27 Haitao Wang. Constructing many faces in arrangements of lines and segments. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3168–3180. SIAM, 2022. doi:10.1137/1.9781611977073.123.

Generating All Invertible Matrices by Row Operations

Petr Gregor ✉ 🏠

Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Hung P. Hoang ✉ 🏠

Algorithm and Complexity Group, Faculty of Informatics, TU Wien, Austria

Arturo Merino ✉ 🏠

Institute of Engineering Sciences, Universidad de O'Higgins, Rancagua, Chile

Ondřej Mička ✉ 🏠

Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Abstract

We show that all invertible $n \times n$ matrices over any finite field $\mathbb{F}_q$ can be generated in a *Gray code* fashion. More specifically, there exists a listing such that (1) each matrix appears exactly once, and (2) two consecutive matrices differ by adding or subtracting one row from a previous or subsequent row, or by multiplying or dividing a row by the generator of the multiplicative group of $\mathbb{F}_q$. This even holds in the more general setting where the pairs of rows that can be added or subtracted are specified by an arbitrary transition tree that has to satisfy some mild constraints. Moreover, we can prescribe the first and the last matrix if $n \geq 3$, or $n = 2$ and $q > 2$. In other words, the corresponding flip graph on all invertible $n \times n$ matrices over $\mathbb{F}_q$ is Hamilton connected if it is not a cycle. This solves yet another special case of Lovász conjecture on Hamiltonicity of vertex-transitive graphs.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory

Keywords and phrases Hamilton cycle, combinatorial Gray code, invertible matrices, finite field, general linear group, generation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.35

Related Version Full proofs are provided in the preprint version of this extended abstract.
arXiv Version: `https://arxiv.org/abs/2405.01863` [8]

Funding This work was supported by Czech Science Foundation grant GA 22-15272S.
Hung P. Hoang: Austrian Science Foundation (FWF, project Y1329 START-Programm)
Arturo Merino: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

Acknowledgements This work was initiated at the 2nd Combinatorics, Algorithms, and Geometry workshop in Dresden, Germany in 2022. We would like to thank the organizers and participants for the inspiring atmosphere.

1 Introduction

Combinatorial generation is one of the most basic tasks we can perform on combinatorial objects and a key topic in Volume 4A of Knuth's seminal series *The Art of Computer Programming* [11]. In this task, we are given an implicit description of the objects and need to produce a listing of all objects fitting the description, with each object appearing exactly once. The goal is to develop an algorithm that can generate these objects at a fast rate.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 35; pp. 35:1–35:14
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

If consecutive objects produced by a generation algorithm differ by large changes, the algorithm must spend a lot of time updating its data structures. Therefore, a natural first step towards creating an efficient generation algorithm is to ensure that consecutive objects differ by only a *small change*. Such a listing is known as a **(combinatorial) Gray code**; see Mütze's survey [13] for many Gray codes of various objects. In addition to combinatorial generation, Gray codes are also relevant in the field of combinatorial reconfiguration, which examines the relationships between combinatorial objects through their local changes; see, e.g., Nishimura's recent introduction on reconfiguration [14].

In this paper, we study Gray codes for invertible matrices over a finite field. A natural attempt for enumerating all invertible $n \times n$ matrices over a finite field $\mathbb{F}_q$, is to choose any nonzero first row and then selecting the following rows to be independent to the previous rows. However, this attempt is not efficient as it requires multiple checks for independence to generate even a single matrix. Furthermore, consecutive matrices in this listing may differ in multiple rows. Instead, we focus on generating matrices in a Gray code order, i.e., every matrix is obtained from the previous one by a single elementary row operation. We note that generating invertible matrices with specific properties has applications in cryptography, e.g., in McEliece cryptosystems [7].

1.1 Strong Lovász conjecture

All invertible $n \times n$ matrices over $\mathbb{F}_q$ with matrix multiplication form the **general linear group** $GL(n, q)$. Each elementary row operation can be represented by multiplying on the left by a matrix that corresponds to this row operation. Hence, we are interested in finding a Hamilton path in an (undirected) Cayley graph on $GL(n, q)$ generated by the allowed row operations, which is in turn an instance of Lovász conjecture [12] on the Hamiltonicity of vertex-transitive graphs.[1]

Stronger versions of Lovász conjecture have been considered in the literature. For example, Dupuis and Wagon [6] asked which non-bipartite vertex-transitive graphs are not Hamilton connected. A graph is **Hamilton connected** if there is a Hamilton path between any two vertices. Similarly, they asked which bipartite vertex-transitive graphs are not Hamilton laceable [6]. A bipartite graph is **Hamilton laceable** if there is a Hamilton path between any two vertices from different bipartite sets. Note that the bipartite sets must be of equal size, which is true for all vertex-transitive bipartite graphs except K_1.

▶ **Conjecture 1** (Strong Lovász conjecture). *For every finite connected vertex-transitive graph G it holds that G is Hamilton connected, or Hamilton laceable, or a cycle, or one of the five known counterexamples.*

The five known counterexamples are the dodecahedron graph, the Petersen graph, the Coxeter graph, and the graphs obtained from the latter two by replacing each vertex with a triangle. The dodecahedron graph is a non-bipartite vertex-transitive graph that has a Hamilton cycle, but it is not Hamilton connected [6]. The other four well-known counterexamples are non-bipartite vertex-transitive graphs that do not admit a Hamilton cycle. Note that except when $G \in \{K_1, K_2, C_3, C_4\}$ the cases in the conjecture are mutually exclusive.

There are many results in line with Conjecture 1. Particularly relevant to us is a result of Tchuente [18] showing that the Cayley graph of the symmetric group S_n, generated by any connected set of transpositions, is Hamilton laceable when $n \geq 4$. Another relevant example is

[1] A graph is **vertex-transitive** if its automorphism group acts transitively on the vertices.

Chen and Quimpo's Theorem [4] showing that all Abelian Cayley graphs satisfy Conjecture 1. Nevertheless, Conjecture 1 remains open even for Cayley graphs of the symmetric group with every generator an involution [16]. Note that none of the five counterexamples to Conjecture 1 is a Cayley graph, leading to Cayley graph variants of Conjecture 1 (e.g., [15]).

1.2 Row operations

Our aim when generating all invertible matrices by row operations is to restrict the allowed operations as much as possible. Note that for $q > 2$ we must allow row multiplications by some scalar to be able to generate all 1×1 matrices. Thus, we allow row multiplications by a fixed generator α of the multiplicative group of nonzero elements of $\mathbb{F}_q$. We will also allow row multiplication by α^{-1}; i.e., division by α, to have an inverse operation for an undirected version of the problem. Furthermore, we specify allowed row additions and subtractions by a directed **transition graph** T on the vertex set $[n]$, where $[n] := \{1, \ldots, n\}$. An edge $(i, j) \in E(T)$ specifies that we can add to or subtract from the j-th row the i-th row. Then, each allowed row operation above corresponds to the left multiplication by a corresponding matrix from a set $\mathrm{ops}(T)$, formally defined by (2).

Observe that to generate all invertible matrices by the allowed operations, the transition graph T *must* be strongly connected; see Lemma 3 below. For our main result we require the following stronger condition.

▶ **Definition 1** (Bypass transition graph). A transition graph T on the vertex set $[n]$ is a **bypass transition graph** if either (i) $n = 1$, or (ii) $n \geq 2$ and

- there exist an edge (i, n) and an edge (n, j) for some $i, j \in [n-1]$, and
- the graph $T - n$ obtained by removing n from T is also a bypass transition graph.

In other words, a bypass transition graph is obtained from a single vertex 1 by repeatedly adding a directed path (a '*bypass*') from some vertex i to some vertex j via a new vertex n. An example of a transition graph with the above property is the one comprised by edges $(i, i+1)$ and $(i+1, i)$ for all $i \in [n-1]$; i.e., a bidirectional path. In the language of row operations, this corresponds to allowing row additions or subtractions between any two consecutive rows. It can be easily seen by induction that a bypass transition graph is strongly connected.

1.3 Our results

For any integer $n \geq 1$, a finite field $\mathbb{F}_q$, and an n-vertex transition graph T we define the (undirected) Cayley graph

$$G(n, q, T) := \mathrm{Cay}(GL(n, q), \mathrm{ops}(T)),$$

where the set $\mathrm{ops}(T)$ is given by (2). Our main result is as follows.

▶ **Theorem 1.** *Let $n \geq 2$ be an integer and q be a prime power such that $q \geq 3$ if $n = 2$. Let T be an n-vertex bypass transition graph. Then the graph $G(n, q, T)$ is Hamilton connected.*

Note that for $n = 1$ the transition graph T has no edges, so $G(1, q, T)$ for any $q \geq 3$ is simply a $(q-1)$-cycle and $G(1, 2, T) = K_1$. For $n = q = 2$, we have that $T = (\{1, 2\}, \{(1, 2), (2, 1)\})$ is the only bypass transition graph, so $G(n, q, T)$ is a 6-cycle, which is not Hamilton connected. Thus, we may restate our result as follows.

▶ **Corollary 2.** *Let $n \geq 1$ be an integer and q be a prime power, and let T be an n-vertex bypass transition graph. Then the graph $G(n, q, T)$ is Hamilton connected unless it is a cycle.*

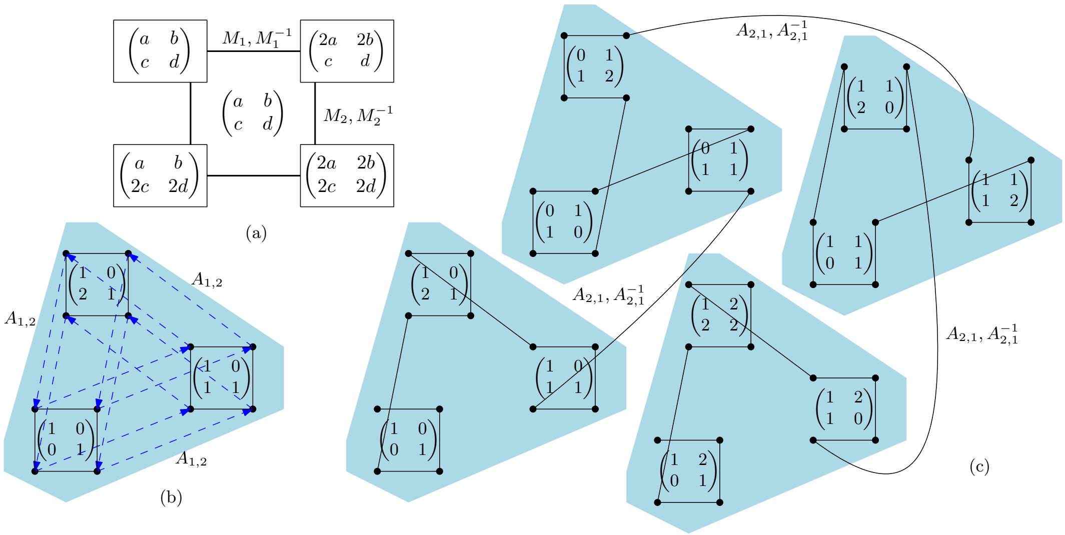


Figure 1 The part (c) illustrates a Hamilton path in the graph $G(2, 3, ([2], \{(1, 2), (2, 1)\}))$. Four vertices around a matrix Z are those obtained from Z by multiplying or dividing a row by α (which is 2 for $q = 3$); see the part (a). The part (b) shows the edges within a shaded component, where the black solid edges are row multiplications/divisions, while the (directed) dashed edges are additions from the first row to the second row. Note that the other directions of the latter edges indicate subtractions of the first row from the second row. Furthermore, while these shaded components exhibit a Cartesian product structure, the same does not hold for the whole graph.

This shows that the family of graphs $G(n, q, T)$ where T is a bypass transition graph is yet another example of a family of Cayley graphs satisfying Conjecture 1. A particularly interesting example is when T is a bidirectional path. See Figure 1 for an illustration for $n = 2$ and $q = 3$.

Moreover, we discuss how to turn the proof of Theorem 1 algorithmic in Section 7.

1.4 Related work

Permutations of $[n]$ can be represented as (invertible binary) permutation matrices forming a subgroup of $GL(n, 2)$. Thus, all the vast results on generating permutations such as in [17, 18] can be directly translated into the context of generating permutation matrices. In particular, there is a general permutation framework developed in [10] that allows us to generate many combinatorial classes by encoding them into permutations avoiding particular patterns. However, the row operations that we consider here do not preserve the subgroup of permutation matrices, so our results do not fall into this framework.

A related task to generation is random sampling. The construction of a random invertible $n \times n$ matrix over $\mathbb{F}_q$ is usually done by constructing a uniformly random matrix and checking whether it is non-singular. The success probability is lower-bounded by a constant independent of n but dependent on q (e.g., see [5] and the citations therein). Hence, there is only a constant factor overhead for random sampling of an invertible matrix over a finite field compared to that of a matrix over the same field. The latter task can be achieved, for example, by independently constructing each row (or column).

2 Preliminaries

The **(undirected) Cayley graph** of a group Γ with a generator set S is the graph $\mathrm{Cay}(\Gamma, S) := (\Gamma, \{\{x, sx\} : x \in \Gamma, s \in S\})$, assuming that S is closed under inverses and does not contain the neutral element. Note that we apply generators on the left as it is more natural for row operations on matrices.

The **general linear group** $GL(n,q)$ is the group of all invertible $n \times n$ matrices over the finite field $\mathbb{F}_q$ with matrix multiplication. Note that for $\mathbb{F}_q$ to be a field, q has to be a prime power. For example, $GL(1,2)$ is the trivial group, $GL(2,2) \simeq S_3$, and $GL(3,2) \simeq PSL(2,7)$ is also known as the group of automorphisms of the Fano plane. The number of elements in $GL(n,q)$ is $a_n := (q^n - 1)(q^n - q) \cdots (q^n - q^{n-1})$, which is obtained by counting choices for (nonzero) rows that are not spanned by the previous rows. It also satisfies the recurrence

$$a_n = (q^n - 1)q^{n-1}a_{n-1}, \tag{1}$$

for $n \geq 2$, and $a_1 = q - 1$ (i.e., the number of nonzero elements of $\mathbb{F}_q$).

By Gaussian elimination, the group $GL(n,q)$ can be generated by row additions and row multiplications by a scalar. As we consider the Cayley graph to be undirected, we also consider the inverse operations, which we call row subtractions and row divisions by a scalar. The formal definitions of these operations are as follows.

For $i \in [n] := \{1, \ldots, n\}$, let $r_i = r_i(A)$ denote the i-th row in A. For distinct $x, y \in [n]$, we denote by $A_{xy} = (a_{ij})$ the binary matrix with $a_{ij} = 1$ if and only if $i = j$, or ($i = y$ and $j = x$). Note that left multiplication by A_{xy} corresponds to adding the x-th row to the y-th row; i.e., the operation $r_x + r_y \to r_y$. Similarly, multiplication by A_{xy}^{-1} then corresponds to subtracting the x-th row to the y-th row; i.e., the operation $-r_x + r_y \to r_y$.

Let α be a generator of the multiplicative group of $\mathbb{F}_q$. For $x \in [n]$, we denote by $M_x = (a_{ij})$ the matrix with $a_{ij} = \alpha$ if $i = j = x$, $a_{ij} = 1$ if $i = j \neq x$, and $a_{ij} = 0$ otherwise. Left multiplication by M_x corresponds to multiplying the x-th row by α; i.e., the operation $\alpha r_x \to r_x$, and multiplication by M_x^{-1} corresponds to the inverse operation $\alpha^{-1} r_x \to r_x$ that we call **dividing** the x-th row by α. Note that for $q = 2$ the multiplicative group is trivial, that is, $M_x = I$, where I denotes the identity matrix.

A **transition graph** T is any directed graph on the vertex set $[n]$ with the edge set $E(T)$. For a transition graph T and a field $\mathbb{F}_q$ we define

$$\mathrm{ops}(T) := \{A_{ij}, A_{ij}^{-1} : (i,j) \in E(T)\} \cup \{M_i, M_i^{-1} : i \in [n]\}, \tag{2}$$

for $q > 2$, and $\mathrm{ops}(T) := \{A_{ij}, A_{ij}^{-1} : (i,j) \in E(T)\}$ for $q = 2$. In other words, $\mathrm{ops}(T)$ contains the row additions and subtractions induced by the edges of T, and all row multiplications and divisions by α if they are nontrivial. A directed graph is strongly connected if for any two vertices i, j, there is a directed path from i to j. A (strongly connected) component of a directed graph is a maximal induced subgraph that is strongly connected. We make the following observation, whose proof can be found in [8].

▶ **Lemma 3.** *For every transition graph T, the set ops(T) generates the group $GL(n,q)$ if and only if T is strongly connected.*

We denote by $\mathbb{F}_q^n$ the vector space of all n-tuples over the field $\mathbb{F}_q$. The span of $u_1, \ldots, u_k \in \mathbb{F}_q^n$ is denoted by $\langle u_1, \ldots, u_k \rangle$. Its orthogonal space $\langle u_1, \ldots, u_k \rangle^\perp$ is the kernel of the matrix with rows $u_1, \ldots, u_k$.

For $k \geq 3$ we denote by C_k a cycle on k vertices, and for $k \in \{1, 2\}$ we define C_k as the complete graph K_k. We also denote the path on k vertices by P_k for $k \geq 1$. The **Cartesian product** $G \,\square\, H$ of two graphs G and H is the graph with the vertex set $V(G) \times V(H)$ and the edge set $\{(u,v)(u',v) : uu' \in E(G), v \in V(H)\} \cup \{(u,v)(u,v') : u \in V(G), vv' \in E(H)\}$. For a graph G and a subset U of vertices, we denote by $G[U]$ the subgraph of G induced by U. Similarly, for a graph G and two subsets of vertices $U_1, U_2 \subseteq V$, we use $E[U_1, U_2]$ to denote the set of edges between U_1 and U_2, i.e., $E[U_1, U_2] = \{xy \in E : x \in U_1, y \in U_2\}$.

For an edge-colored graph, a trail in a graph is **alternating** if any two consecutive edges on the trail differ in color.

3 Joining lemma for Hamilton connectivity

In this section, we present a lemma that joins many Hamilton connected graphs into a larger one. This lemma seems quite versatile. Not only is it useful in our proof in the next section, but it also allows us to easily reprove several classical results on Hamilton connectivity, for example for the permutahedron [18].

▶ **Lemma 4** (Joining lemma). *Let G be a graph with the vertex set partitioned into $k \geq 2$ disjoint subsets $V_1, \ldots, V_k$ such that following conditions hold.*

(1) *$G[V_i]$ is Hamilton connected for every $i \in [k]$;*

(2) *Every vertex in every set V_i has a neighbor in some different set V_j;*

(3) *There are at least three pairwise disjoint edges between every two sets V_i, V_j.*[2]

Then G is Hamilton connected.

Proof. Let $x, y \in V$ be two vertices to be connected by a Hamilton path. First we consider the case when they are in different sets V_i. We can assume that $x \in V_1$ and $y \in V_k$, otherwise we rename the sets. We select vertices $x_i, y_i \in V_i$ for every $i \in [k]$ so that $x_1 = x$, $y_k = y$, $x_i \neq y_i$ for every $i \in [k]$, and y_i is a neighbor of x_{i+1} for every $i \in [k-1]$. Such vertices exist since there are at least three edges between V_i and V_{i+1} for every $i \in [k-1]$ by the condition (3). Then we concatenate Hamilton paths in $G[V_i]$ between x_i and y_i for each $i = 1, \ldots, k$ that exist by the condition (1) into a Hamilton xy-path in G. Note that in this case we did not use the condition (2).

In the second case x and y are in the same set V_i. We can assume that $x, y \in V_1$. Let P be a Hamilton path in $G[V_1]$ between x and y. If $k = 2$, let ab be an edge of P such that the neighbors a' and b' of a and b in V_2, respectively, are distinct. Such neighbors exist by the condition (2), and such an edge ab exists, because otherwise all vertices in V_1 are only adjacent to one vertex in V_2, a contradiction to the condition (3). By replacing the edge ab with the edge aa', a Hamilton path of $G[V_2]$ between a' and b', and the edge $b'b$ we obtain a Hamilton xy-path in G.

If $k > 2$, let ab be an edge of P such that a and b have neighbors a' and b', respectively, in different sets V_i for $i > 1$. Such an edge ab exists since every vertex of V_1 has a neighbor in some other set V_i by the condition (2), and they cannot be all from the same set, say V_2, for otherwise, the condition (3) for the sets V_1 and V_3 would not hold. By the same argument as in the first case, there exists a Hamilton path R between a' and b' in the subgraph $G[V_2 \cup \cdots \cup V_k]$. Finally, replacing the edge ab on P with the edge aa', the path R, and the edge $b'b$ yields a Hamilton xy-path in G. ◀

[2] We could weaken the condition (3) for $k \geq 4$ so that we only need two disjoint edges between all pairs of sets except for two disjoint pairs.

4 Proof of Theorem 1

We will prove the theorem by induction on n, and let $G_n := G(n, q, T)$. We say that an edge $\{X, A_{ij}X\}$ of G_n is **labeled** ij and an edge $\{X, M_iX\}$ of G_n is **labeled** i.

The proof of the base cases for $q = 2$ and $n = 3$, and for $q \geq 3$ and $n = 2$ is deferred to Section 5; see Lemmas 5 and 6. Here, we prove the inductive step, so we assume that $q = 2$ and $n \geq 4$, or $q \geq 3$ and $n \geq 3$, and that the statement holds for the graph $G(n-1, q, T-n)$. Our main tool is the joining lemma from the previous section (Lemma 4).

Proof of the inductive step. We view rows of an invertible $n \times n$ matrix A as an ordered basis $(r_1, \ldots, r_n)$ of the vector space $\mathbb{F}_q^n$. The first $n-1$ rows span a subspace of dimension $n-1$ which is orthogonal to some subspace of dimension 1. That is,

$$\langle r_1, \ldots, r_{n-1} \rangle = \langle u \rangle^{\perp}$$

for a nonzero $u \in \mathbb{F}_q^n$ that satisfies $Ru^T = \mathbf{0}$, where R is the $(n-1) \times n$ matrix whose rows are $r_1, \ldots, r_{n-1}$.

We denote by S_u the set of all the $(n-1) \times n$ matrices whose rows form a basis of $\langle u \rangle^{\perp}$. Observe that this operation gives a bijection between S_u and $GL(n-1, q)$: Remove the i-th column of every matrix in S_u, where i is the index of the first nonzero element of u (which exists, as u is not the zero vector). Furthermore, for every matrix X in S_u, we can add any vector as the last row to form an $n \times n$ invertible matrix, as long as this added vector is independent of the rows of X (i.e., any vector in $\mathbb{F}_q^n \setminus \langle u \rangle^{\perp}$).

Note that this tallies with the count in (1). Recall that a_{n-1} denotes the number of elements in $GL(n-1, q)$. There are $(q^n - 1)/(q-1)$ choices for the one-dimensional subspace $\langle u \rangle$. For each subspace (with a representative basis u), S_u has a_{n-1} elements, due to the aforementioned bijection. Lastly, for each matrix X in S_u, there are $q^{n-1}(q-1)$ possible last rows, which can be obtained by adding a linear combination of the rows of X to an initial last row and then multiplying the sum by a power of α. Together, we recover the recurrence statement (1).

Following the above analysis, we prove the inductive step in four smaller steps.

First, given a one dimensional subspace with a basis u and a vector v independent of the rows of any matrix in S_u, we denote by the tuple (S_u, v) the set of all matrices in $GL(n, q)$ formed by adding v as the last row to each of the matrices in S_u. Since the aforementioned bijection is an isomorphism of $G_n[(S_u, v)]$ and $G(n-1, q, T-n)$, by inductive hypothesis we conclude that $G_n[(S_u, v)]$ is Hamilton connected.

Before we proceed, we note that row multiplications, divisions, and row additions that do not involve the last row only transform a matrix into another matrix in the same set (S_u, v) for some u, v. Next, adding a row to the last row transforms a matrix in (S_u, v) into another matrix in (S_u, v') for $v \neq v'$. Lastly, adding the last row to another row transforms a matrix in (S_u, v) into another matrix in $(S_{u'}, v)$, where $\langle u \rangle \neq \langle u' \rangle$.

Second, we denote by $(S_u, \langle v \rangle)$ the set of all matrices in $GL(n, q)$ formed by adding any multiple of v as the last row to each of the matrices in S_u. Since multiplication by α generates all nonzero elements of $\mathbb{F}_q$, the edges of label n that multiply the last row form a cycle of length $q-1$. Hence, the graph $G_n[(S_u, \langle v \rangle)] \simeq G_n[(S_u, v)] \,\square\, C_{q-1}$, which can be easily showed to be Hamilton connected (see [8]).

Third, given a one dimensional subspace with a basis u, we denote by $(S_u, *)$ the set of all matrices in $GL(n, q)$ whose first $n-1$ rows form a matrix in S_u. Here, we use Lemma 4 to join the subgraphs $G_n[(S_u, \langle v \rangle)]$ for all applicable v to prove the Hamilton connectivity of $G_n[(S_u, *)]$. The joining edges between these subgraphs have label in for $i \in [n-1]$ (such an i is guaranteed by the bypass property of T). In order to use the lemma, we show that all of its conditions hold. The condition (1) follows the second step above. The condition (2) is satisfied, because for every $u, v \in \mathbb{F}_q^n$ and a matrix X in $(S_u, v) \subseteq (S_u, \langle v \rangle)$, $A_{in}X$ is a neighbor of X in G_n and in $(S_u, \langle v + r_i(X) \rangle)$, a different set than $(S_u, \langle v \rangle)$. For the condition (3), given u and two distinct $v, v' \notin \langle u \rangle^\perp$ such that $\langle v \rangle \neq \langle v' \rangle$, we have that v is a linear combination of a basis of $\langle u \rangle^\perp$ and v', and consequently $v = x + av'$ for some nonzero $x \in \langle u \rangle^\perp$ and a nonzero $a \in \mathbb{F}_q$. As S_u contains all matrices whose rows form a basis in $\langle u \rangle^\perp$, there exist three matrices X_1, X_2, and X_3 in S_u such that their i-th row is $x = v - av'$. We can guarantee three matrices in S_u, because in the inductive step, $n \geq 3$ and $q \geq 3$, or $n \geq 4$ and $q = 2$, and hence, when we fix the $n - 2$ rows including the i-th row, there are at least three different choices for the remaining row. Then the edges $\{X_1, A_{in}X_1\}$, $\{X_2, A_{in}X_2\}$, and $\{X_3, A_{in}X_3\}$ are the three distinct edges as required by the condition (3). We can now apply Lemma 4 and conclude that $G_n[(S_u, *)]$ is Hamilton connected.

Last, we again apply Lemma 4 to join the different subgraphs $G_n[(S_u, *)]$ for all subspaces $\langle u \rangle$ to complete the inductive step. Here, the joining edges have the label nj for some $j \in [n-1]$, which exist because T is a bypass transition graph. The condition (1) of the lemma follows the previous step. The condition (2) is satisfied, because for any X in some $(S_u, *)$, $A_{nj}X$ is a neighbor of X in G_n and belongs to a different set $(S_{u'}, *)$. For the condition (3), given u, u' not in the same one-dimensional subspace, $\langle u, u' \rangle^\perp$ is a subspace of dimension $n - 2$.

If $n \geq 4$, or $n = 3$ and $q > 3$, there exist three distinct matrices B, B', and B'' whose rows form bases of this $(n-2)$-dimensional subspace. The remaining case $n = 3$ and $q = 3$ is considered separately below. Let $v_u \in \langle u \rangle^\perp \setminus \langle u' \rangle^\perp$ and $v_{u'} \in \langle u' \rangle^\perp \setminus \langle u \rangle^\perp$. Clearly, we have that $v_{u'} - v_u$ is independent of the rows of each matrix B, B', and B''. Let $\tilde{B}$, $\tilde{B}'$, and $\tilde{B}''$ be the $n \times n$ matrix obtained from B, B', and B'' respectively by inserting a new row v_u at the j-th position and $v_{u'} - v_u$ as the last row.

If $n = 3$ and $q = 3$ we have $\langle u \rangle^\perp \cap \langle u' \rangle^\perp = \langle w \rangle = \{0, w, 2w\}$ for some nonzero $w \in \mathbb{F}_3^3$, so there are only two distinct matrices whose rows form bases of this 1-dimensional subspace, in particular $B = (w)$ and $B' = (2w)$. In this case, we define $\tilde{B}$ and $\tilde{B}'$ as in the previous case, but for $\tilde{B}''$ we take the matrix obtained from B by inserting a new row $2v_u$ at the j-th position and $2v_{u'} - 2v_u$ as the last row.

In both cases, $\{\tilde{B}, A_{nj}\tilde{B}\}$, $\{\tilde{B}', A_{nj}\tilde{B}'\}$, and $\{\tilde{B}'', A_{nj}\tilde{B}''\}$ are the three edges as required by the condition (3). This completes all the conditions of Lemma 4 and completes the inductive step. ◀

5 Base cases for the induction

We verify the case when $n = 3$ and $q = 2$ by computer search in SageMath; see [8].

▶ **Lemma 5.** *For every bypass transition graph T, the graph $G(3, 2, T)$ is Hamilton connected.*

Since the only bypass transition graph T for $n = 2$ is the complete graph $T = ([2], \{(1, 2), (2, 1)\})$, the remaining base cases for $q \geq 3$ and $n = 2$ are captured in the following lemma.

▶ **Lemma 6.** *For a prime power $q \geq 3$, $G(2, q, ([2], \{(1, 2), (2, 1)\}))$ is Hamilton connected.*

To prove Lemma 6, we first consider the graph that arises by removing the edges that add the second row to the first row; i.e., for a prime power $q \in \mathbb{N}$, we define $G'(q) := G(2, q, ([2], \{(1, 2)\}))$. The graph $G'(q)$ is disconnected, and thus, we consider the connected component in $G'(q)$ that contains the identity matrix and denote it by $H(q)$. We will usually write H and G' for $H(q)$ and $G'(q)$ whenever there is no risk of confusion.

It is easy to see that the vertices of H are of the form:

$$V(H) = \left\{ \begin{pmatrix} \alpha^i & 0 \\ \alpha^j a & \alpha^j \end{pmatrix} : i, j \in \{0, \ldots, q-2\}, a \in \mathbb{F}_q \right\}.$$

Furthermore, the graph H has a simple structure when analyzing the components by fixing $a \in \mathbb{F}_q$, as described in the following. Let us define

$$V_a = \left\{ \begin{pmatrix} \alpha^i & 0 \\ \alpha^j a & \alpha^j \end{pmatrix} : i, j \in \{0, \ldots, q-2\} \right\}$$

and let H_a be the graph induced by fixing a in H; i.e., $H_a := H[V_a]$. We have the following simple observations regarding H and its decomposition by fixing $a \in \mathbb{F}_q$.

(p1) *(H splits into copies of H_a.)* Removing the edges that add the first row to the second row in H disconnects H and splits it into the connected components $\{H_a : a \in \mathbb{F}_q\}$.

(p2) *(The graphs H_a have good Hamiltonicity properties.)* For every $a \in \mathbb{F}_q$, we get that H_a is a toroidal grid of dimensions $(q-1) \times (q-1)$ where each dimension of the grid is given by multiplication by α in the respective row; i.e., $H_a \cong C_{q-1} \square C_{q-1}$. In particular, H_a is isomorphic to H_b for every $a, b \in \mathbb{F}_q$.

(p3) *(The components H_a are well-connected.)* For every $i \in \{0, \ldots, q-2\}$ and $a, b \in \mathbb{F}_q$ such that $a \neq b$, we have that

a. $\alpha^i \begin{pmatrix} a-b & 0 \\ a & 1 \end{pmatrix} \in V_a$ and $\alpha^i \begin{pmatrix} a-b & 0 \\ b & 1 \end{pmatrix} \in V_b$ are connected by an edge,

b. $\alpha^i \begin{pmatrix} b-a & 0 \\ a & 1 \end{pmatrix} \in V_a$ and $\alpha^i \begin{pmatrix} b-a & 0 \\ b & 1 \end{pmatrix} \in V_b$ are connected by an edge,

and no other edges between V_a and V_b exist. In particular, for every $a, b \in \mathbb{F}_q$ such that $a \neq b$ we have that $|E[V_a, V_b]| = 2(q-1)$ with all the edges being disjoint.

We exploit these properties as follows: We split H into the components H_a for $a \in \mathbb{F}_q$, then since the graphs H_a are either Hamilton laceable or connected and the components H_a are well-connected we can glue the corresponding Hamilton paths in each H_a to form a Hamilton path in H.

If q is even, for every $a \in \mathbb{F}_q$ the graphs H_a are Hamilton connected. This makes it easier to lift the Hamilton paths from H_a to a Hamilton path in H. However, when q is odd, the picture is much different. In particular, there are parity constraints given by the fact that for every $a \in \mathbb{F}_q$ the graph H_a is now bipartite. To make this formal, we partition $V(H)$ into two colors. We say that $x = \begin{pmatrix} \alpha^i & 0 \\ \alpha^j a & \alpha^j \end{pmatrix}$ is **blue** whenever $i + j$ is even, and it is **red** whenever $i + j$ is odd. We denote the **color of a vertex** $x \in V(H)$ by $\operatorname{col}(x)$. It is easy to show that if $x \in V_a$, $y \in V_b$ for some $a \neq b$ and there is an edge $xy \in E[V_a, V_b]$, then $\operatorname{col}(x) = \operatorname{col}(y)$. Thus, for every edge $xy \in E[V_a, V_b]$ we can define its **(edge) color** as $\operatorname{col}(xy) := \operatorname{col}(x) = \operatorname{col}(y)$.

If $q \equiv 3 \pmod 4$, a simple computation shows that whenever there is a red or blue edge between V_a and V_b for $a, b \in \mathbb{F}_q$ we also have an edge of the opposite color. Thus, the coloring does not impose any extra restrictions.

The problematic case occurs whenever $q \equiv 1 \pmod 4$. In this case, all the edges between V_a and V_b have the same color for $a, b \in \mathbb{F}_q$. Hence, it is natural to consider the graph where we contract every V_a for $a \in \mathbb{F}_q$. Thus, we obtain a new graph $\bar{K}_q$ with $\mathbb{F}_q$ as vertices, and for the edges $xy \in E[V_a, V_b]$ we put a new edge ab colored with $\operatorname{col}(xy)$. This graph is a complete graph on $\mathbb{F}_q$ where the coloring of the edges can be succinctly described as follows:

(*) For x and y in $\mathbb{F}_q$, the edge xy has color red (blue) if there exists an odd (even) $z \in \mathbb{Z}$, such that $x - y = \alpha^z$. (See Figure 2 for an example.)

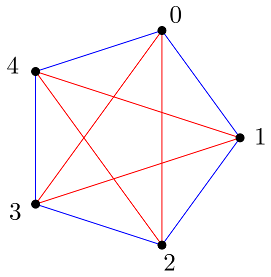


Figure 2 The graph $\bar{K}_5$ for $\alpha = 2$.

If we plan to have a Hamilton path of H that traverses each set V_a at a time for $a \in \mathbb{F}_q$, then for any $a, b \in \mathbb{F}_q$, there is at most one edge of the Hamilton path that crosses between V_a and V_b. Further, as each V_a has even size, this means that as we traverse this Hamilton path, any two consecutive such "crossing" edges have to differ in color. This translates to the requirement that we should have an alternating Hamilton path in $\bar{K}_q$. We show in the next lemma that this holds, even for Hamilton connectivity.

▶ **Lemma 7.** *Let q be a prime power, $q \equiv 1 \pmod 4$. For any two distinct vertices $a, b \in \mathbb{F}_q$ and a color c of either red or blue, there exists an alternating Hamilton ab-path of $\bar{K}_q$ such that a is incident to an edge of color c on the path.*

We defer the proof of Lemma 7 to Section 6. We can use Lemma 7 to prove Hamiltonicity properties of subgraphs of H. To this end, we have the following definition.

▶ **Definition 1.** *An induced subgraph H' of H is* ***structured*** *if and only if the following holds:*

1. *For every $a \in \mathbb{F}_q$ we have that $H'[V_a]$ is isomorphic to either $C_{q-1} \square C_{q-1}$ or $C_{q-1} \square P_\ell$ for $\ell \geq (q-1)/2$, and*
2. *For every distinct $a, b \in \mathbb{F}_q$, there is at least one edge between $H'[V_a]$ and $H'[V_b]$.*

▶ **Lemma 8.** *Let $q \geq 5$ be an odd integer and H' be a structured induced subgraph of $H(q)$. Let $x, y \in V(H(q))$ be two vertices of different colors such that $x \in V_a$, $y \in V_b$ with distinct $a, b \in \mathbb{F}_q$. Then there exists a Hamilton xy-path in H'.*

The reader may notice similarities between Lemma 8 and the joining lemma (Lemma 4). In particular, they may wonder why we require only *one* edge between components, instead of the three needed in the joining lemma. Recall that the need for three edges in the joining lemma was in the case where we want to have an xy-subpath that spans two consecutive components, but the edges that cross between these two components are incident to either x or y. However, this cannot happen for structured graphs, because the coloring conditions force these endpoints not to be used in crossing edges. The proof of this lemma is presented in [8].

Equipped with Lemmas 4 and 8, we can show the following lemma, whose full proof is also presented in [8].

▶ **Lemma 9.** *For every $q \geq 3$ the graph $H(q)$ is Hamilton connected.*

We are now ready to prove Lemma 6.

Proof of Lemma 6. Let us denote $S = \{(a,1) : a \in \mathbb{F}_q\} \cup \{(1,0)\} \subseteq \mathbb{F}_q^2$. For any nonzero vector $u \in S$, let us define $V^u \subseteq V(G)$ to be the set of all matrices in G with the first row in $\langle u \rangle$. Note that each V^u corresponds to a unique component of G', with $V^{(1,0)} = V(H)$.

We apply Lemma 4 with partitioning $\{V_u : u \in S\}$.

We begin by simplifying our arguments using symmetry. Note that any two components of G' are isomorphic via $f_A : x \mapsto xA$ for some $A \in GL(2,q)$. Moreover, for any A the mapping f_A is an automorphism of G that preserves operations on the edges; i.e., if an edge corresponds to the operation M_1, then it will be mapped to some edge that also corresponds to M_1. In particular, H is isomorphic to every other component.

For the condition (1) of Lemma 4, we know that H is Hamilton connected by Lemma 9, so by isomorphism the same holds for every $G[V^u]$. The condition (2) is satisfied simply by adding the second row to the first row. For the condition (3), we first observe that if there is an edge between $G[V^u]$ and $G[V^{u'}]$, there are actually at least $q-1$ disjoint edges as we can multiply both matrices by α^i. By isomorphism, it is enough to show that there is an edge between H and any V^u distinct from $V^{(1,0)}$. Since $u = (a,1)$ for some $a \in \mathbb{F}_q$, we can use the edges between $\begin{pmatrix} 1 & 0 \\ a-1 & 1 \end{pmatrix} \in V(H)$ and $\begin{pmatrix} a & 1 \\ a-1 & 1 \end{pmatrix} \in V^{(a,1)}$. ◀

6 Alternating path in two-edge-colored complete graph

In this section, we prove Lemma 7, which is needed in the proof of Lemma 8.

We start with a brief recap of the context needed for this lemma. Suppose q is a prime power and $q \equiv 1 \pmod 4$. We remind the reader that α is a generator of the multiplicative group of $\mathbb{F}_q$. Recall that $\bar{K}_q$ is the complete graph on the vertex set $\mathbb{F}_q$ with edges colored by the following scheme:

(*) For x and y in $\mathbb{F}_q$, the edge xy has color red (blue) if there exists an odd (even) $z \in \mathbb{Z}$ such that $x - y = \alpha^z$.

Our goal is to find an alternating Hamilton path between two prescribed vertices a and b with a prescribed color of the edge incident to a.

We begin by arguing that $\bar{K}_q$ is well-defined. Let 0 be the additive identity and 1 be the multiplicative identity of $\mathbb{F}_q$. By the definition of α, the nonzero elements of $\mathbb{F}_q$ are exactly $\alpha^0, \dots, \alpha^{q-2}$. Furthermore, $\alpha^i = \alpha^{q-1+i}$ for all integers $i \in \mathbb{Z}$. Since q is odd, we conclude that if $\alpha^z = \alpha^{z'}$ for some z, z', then z and z' have the same parity. Thus, for x and y in $\mathbb{F}_q$, there exists a unique $p \in \{0,1\}$ such that if $x - y = \alpha^z$ then $z \equiv p \pmod 2$. Further, since $\alpha^{(q-1)/2} = -1$, if $x - y = \alpha^z$, then $y - x = \alpha^{z'}$ for $z' = z + (q-1)/2$. As $q \equiv 1 \pmod 4$, z and z' have the same parity. Therefore, the color of each edge of $\bar{K}_q$ is well-defined.

The problem of finding an alternating cycle/path in a graph has a long history and a wide range of applications; see the survey by Bang-Jensen and Gutin [1]. However, the existing results on alternating Hamilton cycles/paths in two-edge-colored complete graphs (e.g., [2, 3]) cannot be readily applied in our setting. Furthermore, we also specify the color of the first edge of the path, which is not guaranteed by these results. Therefore, we provide a direct and constructive proof of an alternating Hamilton path in our special complete graph.

In the following proof, we use the observation that the operation of adding a constant to all vertex labels preserves edge colors since the difference between any two vertices remains the same.

▶ **Lemma 7.** *Let q be a prime power, $q \equiv 1 \pmod 4$. For any two distinct vertices $a, b \in \mathbb{F}_q$ and a color c of either red or blue, there exists an alternating Hamilton ab-path of $\bar{K}_q$ such that a is incident to an edge of color c on the path.*

Proof. For $i \in \{0, \ldots, q-1\}$, define $v_i := \sum_{j=0}^{i} \alpha^j$. Note that $v_{q-2} = 0$, and $v_0 = v_{q-1} = 1$. It is easy to see that $(v_0, \ldots, v_{q-2})$ forms an alternating cycle C in $\bar{K}_q$. The only missing vertex in C is $u := -(\alpha-1)^{-1}$. Indeed, if this vertex is on the cycle, then for some t, we must have $(\alpha^{t+1}-1)(\alpha-1)^{-1} = -(\alpha-1)^{-1}$, which implies $\alpha^{t+1} = 0$, a contradiction with the fact that α generates nonzero elements of $\mathbb{F}_q$.

For any $i \in \{0, \ldots, q-2\}$ we have $v_i - u = (\alpha^{i+1}-1)(\alpha-1)^{-1} + (\alpha-1)^{-1} = \alpha^{i+1}(\alpha-1)^{-1}$. By a similar argument, we have that $v_{i+1} - u = \alpha^{i+2}(\alpha-1)^{-1} = \alpha(v_i - u)$. Thus, by the coloring scheme (*), uv_i and uv_{i+1} have different colors.

▷ Claim. For any vertex v in C, there exists an alternating Hamilton uv-path such that on the path, u is incident to an edge whose color is different from that of uv.

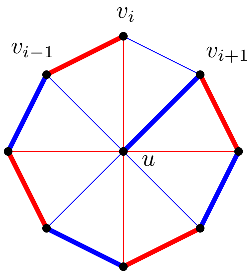


Figure 3 Illustration of the claim's proof. The outer cycle is the cycle C, and the bold edges indicate an alternating Hamilton path.

Proof. Suppose $v = v_i$ for some $i \in \{0, \ldots, q-2\}$. By the argument above, uv_{i-1} and uv_{i+1} have the same color. Further, since C is an alternating cycle, $v_{i-1}v_i$ and v_iv_{i+1} have different colors. Hence, one of these two edges have the same color as uv_{i-1} and uv_{i+1}. Without loss of generality, suppose this edge is v_iv_{i+1}. Then we have $(u, v_{i+1}, v_{i+2} \ldots, v_{q-2}, v_0, \ldots, v_{i-1}, v_i)$ is the desired alternating Hamilton path. See Figure 3 for an illustration. ◁

Consider adding $b-u$ to all vertex labels. The missing vertex from the cycle C above is now b. By the claim above, we obtain an alternating Hamilton ba-path such that the incident edge to b has different color than that of ba. Since q is odd, this implies that the edge incident to a on this path has the same color as ba. Next, we add $a-u$ to all original vertex labels. The missing vertex from C is now a. Again by the claim above, we obtain another alternating Hamilton ab-path such that the incident edge to a has different color than that of ab.

Since the two alternating Hamilton paths above have different colors for the edge incident to a, the lemma follows. ◀

7 Algorithmization

The proof of Theorem 1 can be easily turned into an algorithm that computes a Hamilton path in $G(n, q, T)$ running in time polynomial in $|GL(n, q)|$. This can be obtained by a straightforward recursion based on the joining lemma. More specifcally, the main idea of the proof is to split the graph and proceed recursively. Close examination of the proof of the joining lemma and its applications along our proof shows that such a recursion can be computed in time polynomial in $|GL(n, q)|$; we omit the details.

However, the typical goal from a generation perspective is to have an algorithm that outputs objects one by one with a small delay and preprocessing time. Here, the **delay** is the worst-case time the algorithm takes between consecutively generated objects and the **preprocessing time** is the time before generating any objects. Thus, the natural objective from generation perspective for invertible matrices is an enumeration algorithm running in delay $\mathrm{poly}(n, q) := n^{\mathcal{O}(1)}q^{\mathcal{O}(1)}$ with $\mathrm{poly}(n, q)$ preprocessing. Note that such an algorithm immediately gives a solution to computing Hamilton paths in $G(n, q, T)$ in time $\mathrm{poly}(n, q)|GL(n, q)|$.

The naive implementation of our inductive proof uses a call stack that needs space exponential in n and takes exponential time in n to put the recursive calls in the stack. Despite that, it is still possible to obtain a polynomial delay algorithm by following the recursive structure of the main proof. As highlighted in the stack approach, we cannot store *all* the information given by the recursion. Instead, we only store information related to the *current path* in the recursion tree. More specifically, if we are at a vertex z, we trace back the ℓ recursive calls, each utilizing the joining lemma. The i-th call indicates a pair of a source x_i and a target y_i for which we traverse a Hamilton path. For every $i \in [\ell]$ we store x_i, y_i and a small amount of extra bits serving as a compressed history. It turns out that this is enough information to reconstruct the path of z in the recursion tree and decide how to proceed; more details are given in [8].

8 Open questions

We conclude with several remarks and open questions.

1. **Non-bypass transition graphs.** Does Theorem 1 hold for any strongly connected (and not necessarily bypass) transition graph, in particular for the directed n-cycle? We verified by computer that the result holds for the directed cycle if $q = 2$ and $n = 3$.
2. **Other generators.** Does Theorem 1 hold for other generators of the group $GL(n, q)$? For example, there is the generator $\{M_2A_{1,n}, P_{2\dots n1}\}$ of size 2, where $P_{2\dots n1}$ refers to the permutation matrix corresponding to the permutation $2 \dots n1$ [19]. This problem is similar to the sigma-tau problem for permutations solved by Sawada and Williams [17].
3. **Subgroups of $GL(n, q)$.** As an intermediate step, we show that Cayley graphs of certain subgroups of $GL(n, q)$ are Hamilton connected. Can we prove it for other subgroups that correspond to given restrictions of matrices?
4. **Symmetric Hamilton cycles.** Instead of Hamilton connectivity we may ask for Hamilton cycles that are preserved under a large cyclic subgroup of automorphisms. This problem was recently studied by Gregor, Merino, and Mütze [9] for several highly symmetric graphs. The graphs considered here are also highly symmetric.
5. **Matrices over rings.** Another natural extension is to explore if our results extend to invertible matrices in the ring setting. This is particularly interesting for cyclic rings; i.e., checking Hamiltonicity of Cayley graphs of invertible matrices in $\mathbb{Z}_k$ for $k \in \mathbb{N}$. Naturally, the methods will highly depend on the chosen generators for which there does not seem to be an obvious choice.
6. **Alternating Hamilton paths in 2-colored K_{2n+1}.** Despite our efforts and many existing results on properly colored Hamilton cycles in complete graphs (see a survey [1]), we did not find an answer to the following question. Is it true that the complete graph K_{2n+1} with 2-colored edges so that every vertex is incident with exactly n edges of each color contains an alternating Hamilton path between any two vertices?
7. **Efficient algorithms.** Is there a generating algorithm that achieves $\mathcal{O}(n)$ delay? Is there a simple greedy algorithm?

References

1 J. Bang-Jensen and G. Gutin. Alternating cycles and paths in edge-coloured multigraphs: a survey. *Discrete Math.*, 165/166:39–60, 1997. doi:10.1016/S0012-365X(96)00160-4.

2 J. Bang-Jensen, G. Gutin, and A. Yeo. Properly coloured Hamiltonian paths in edge-coloured complete graphs. *Discrete Appl. Math.*, 82(1-3):247–250, 1998. doi:10.1016/S0166-218X(97)00062-0.

3 M. Bánkfalvi and Zs. Bánkfalvi. Alternating Hamiltonian circuit in two-coloured complete graphs. In *Theory of Graphs (Proc. Colloq., Tihany, 1966)*, pages 11–18. Academic Press, New York-London, 1968.

4 C. C. Chen and N. F. Quimpo. On strongly Hamiltonian abelian group graphs. In *Combinatorial mathematics, VIII (Geelong, 1980)*, volume 884 of *Lecture Notes in Math.*, pages 23–34. Springer, Berlin-New York, 1981. doi:10.1007/BFb0091805.

5 C. Cooper. On the rank of random matrices. *Random Structures Algorithms*, 16(2):209–232, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1.

6 M. Dupuis and S. Wagon. Laceable knights. *Ars Math. Contemp.*, 9:115–124, 2015. doi:10.26493/1855-3974.420.3C5.

7 T. Fabšič, O. Grošek, K. Nemoga, and P. Zajac. On generating invertible circulant binary matrices with a prescribed number of ones. *Cryptogr. Commun.*, 10(1):159–175, 2018. doi:10.1007/s12095-017-0239-4.

8 P. Gregor, H. P. Hoang, A. Merino, and O. Mička. Generating all invertible matrices by row operations. *arXiv preprint*, 2024. Full preprint version of the present article. arXiv:2405.01863.

9 P. Gregor, A. Merino, and T. Mütze. The Hamilton Compression of Highly Symmetric Graphs. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.54.

10 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Trans. Amer. Math. Soc.*, 375(4):2255–2291, 2022. doi:10.1090/tran/8199.

11 D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011.

12 L. Lovász. Problem 11. In *Combinatorial Structures and Their Applications (Proc. Calgary Internat. Conf., Calgary, AB, 1969)*. Gordon and Breach, New York, 1970.

13 T. Mütze. Combinatorial Gray codes—an updated survey. *Electron. J. Combin.*, Dynamic Surveys DS26:93 pp., 2023. doi:10.37236/11023.

14 Naomi Nishimura. Reasons to fall (more) in love with combinatorial reconfiguration. In *WALCOM: algorithms and computation*, volume 14549 of *Lecture Notes in Comput. Sci.*, pages 9–14. Springer, Singapore, 2024. doi:10.1007/978-981-97-0566-5_2.

15 David J. Rasmussen and Carla D. Savage. Hamilton-connected derangement graphs on S_n. *Discrete Math.*, 133(1-3):217–223, 1994. doi:10.1016/0012-365X(94)90028-0.

16 Frank Ruskey and Carla Savage. Hamilton cycles that extend transposition matchings in Cayley graphs of S_n. *SIAM J. Discrete Math.*, 6(1):152–166, 1993. doi:10.1137/0406012.

17 J. Sawada and A. Williams. Solving the sigma-tau problem. *ACM Trans. Algorithms*, 16(1):Art. 11, 17 pp., 2020. doi:10.1145/3359589.

18 M. Tchuente. Generation of permutations by graphical exchanges. *Ars Combin.*, 14:115–122, 1982.

19 W. C. Waterhouse. Two generators for the general linear groups over finite fields. *Linear and Multilinear Algebra*, 24(4):227–230, 1989. doi:10.1080/03081088908817916.

Kernelization Complexity of Solution Discovery Problems

Mario Grobler
University of Bremen, Germany

Stephanie Maaz
University of Waterloo, Canada

Amer E. Mouawad
American University of Beirut, Lebanon

Naomi Nishimura
University of Waterloo, Canada

Vijayaragunathan Ramamoorthi
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Abstract

In the solution discovery variant of a vertex (edge) subset problem Π on graphs, we are given an initial configuration of tokens on the vertices (edges) of an input graph G together with a budget b. The question is whether we can transform this configuration into a feasible solution of Π on G with at most b modification steps. We consider the token sliding variant of the solution discovery framework, where each modification step consists of sliding a token to an adjacent vertex (edge). The framework of solution discovery was recently introduced by Fellows et al. [ECAI 2023] and for many solution discovery problems the classical as well as the parameterized complexity has been established. In this work, we study the kernelization complexity of the solution discovery variants of Vertex Cover, Independent Set, Dominating Set, Shortest Path, Matching, and Vertex Cut with respect to the parameters number of tokens k, discovery budget b, as well as structural parameters such as pathwidth.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Mathematics of computing → Combinatorics

Keywords and phrases solution discovery, kernelization, cut, independent set, vertex cover, dominating set

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.36

Related Version *Full Version*: `https://arxiv.org/abs/2409.17250`

Funding *Naomi Nishimura, Stephanie Maaz*: Research supported by the Natural Sciences and Engineering Research Council of Canada.
Vijayaragunathan Ramamoorthi: Funded by the "Mind, Media, Machines" high-profile area at the University of Bremen.

1 Introduction

In the realm of optimization, traditional approaches revolve around computing optimal solutions to problem instances from scratch. However, many practical scenarios can be formulated as the construction of a feasible solution from an infeasible starting state. Examples of such scenarios include reactive systems involving human interactions. The inherent dynamics of such a system is likely to lead to an infeasible state. However, computing a

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 36; pp. 36:1–36:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solution from scratch may lead to a solution that may differ arbitrarily from the starting state. The modifications required to reach such a solution from the starting state may be costly, difficult to implement, or sometimes unacceptable.

Let us examine a specific example to illustrate. A set of workers is assigned tasks so that every task is handled by a qualified worker. This scenario corresponds to the classical matching problem in bipartite graphs. Suppose one of the workers is now no longer available (e. g., due to illness); hence, the schedule has to be changed. An optimal new matching could be efficiently recomputed from scratch, but it is desirable to find one that is as close to the original one as possible, so that most of the workers keep working on the task that they were initially assigned.

Such applications can be conveniently modeled using the *solution discovery* framework, which is the central focus of this work. In this framework, rather than simply finding a feasible solution to an instance $\mathcal{I}$ of a source problem Π, we investigate whether it is possible to transform a given infeasible configuration into a feasible one by applying a limited number of transformation steps. In this work we consider vertex (edge) subset problems Π on graphs, where the *configurations* of the problem are sets of vertices (edges). These configurations are represented by the placement of tokens on the vertices (edges) of the configuration. An atomic *modification step* consists of moving one of the tokens and the question is whether a feasible configuration is reachable after at most b modification steps. Inspired by the well-established framework of combinatorial reconfiguration [4, 16, 15], commonly allowed modification steps are the addition/removal of a single token, the jumping of a token to an arbitrary vertex/edge, or the slide of a token to an adjacent vertex (edge).

Problems defined in the solution discovery framework are useful and have been appearing in recent literature. Fellows et al. [11] introduced the term *solution discovery*, and along with Grobler et al. [13] initiated the study of the (parameterized) complexity of solution discovery problems for various NP-complete source problems including Vertex Cover (VC), Independent Set (IS), Dominating Set (DS), and Coloring (Col) as well as various source problems in P such as Spanning Tree (ST), Shortest Path (SP), Matching (Mat), and Vertex Cut (VCut) / Edge Cut (ECut).

Fellows et al. [11] and Grobler et al. [13] provided a full classification of polynomial-time solvability vs. NP-completeness of the above problems in all token movement models (token addition/removal, token jumping, and token sliding). For the NP-complete solution discovery problems, they provided a classification of fixed-parameter tractability vs. W[1]-hardness. Recall that a *fixed-parameter tractable algorithm* for a problem Π with respect to a parameter p is one that solves Π in time $f(p) \cdot n^{\mathcal{O}(1)}$, where n is the size of the instance and f is a computable function dependent solely on p, while W[1]-hardness provides strong evidence that the problem is likely not fixed-parameter tractable (i. e., does not admit a fixed-parameter tractable algorithm) [9].

A classical result in parameterized complexity theory is that every problem Π that admits a fixed-parameter tractable algorithm necessarily admits a kernelization algorithm as well [5]. A *kernelization algorithm* for a problem Π is a polynomial-time preprocessing algorithm that, given an instance x of the problem Π with parameter p, produces a *kernel* – an equivalent instance x' of the problem Π with a parameter p', where both the size of x' and the parameter p' are bounded by a computable function depending only on p [9]. Typically, kernelization algorithms generated using the techniques of Cai et al. [5] yield kernels of exponential (or even worse) size. In contrast, designing problem-specific kernelization algorithms frequently yields more efficiently-sized kernels, often quadratic or even linear with respect to the parameter. Note that once a decidable problem Π with parameter p

admits a kernelization algorithm, it also admits a fixed-parameter tractable algorithm, as a kernelization algorithm always produces a kernel of size that is simply a function of p. The fixed-parameter tractable solution discovery algorithms of Fellows et al. [11] and Grobler et al. [13] are not based on kernelization algorithms.

Unfortunately, it is unlikely that all fixed-parameter tractable problems admit polynomial kernels. Bodlaender et al. [2, 3] developed the first framework for proving kernel lower bounds and Fortnow and Santhanam [12] showed a connection to the hypothesis $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. Specifically, for several NP-hard problems, a kernel of polynomial size with respect to a parameter would imply that $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, and thus an unlikely collapse of the polynomial hierarchy to its third level [18]. Driven by the practical benefits of kernelization algorithms, we explore the size bounds on kernels for most of the above-mentioned solution discovery problems in the token sliding model, particularly those identified as fixed-parameter tractable in the works of Fellows et al. [11] and Grobler et al. [13].

1.1 Results Overview

We focus on the kernelization complexity of solution discovery in the token sliding model for the following source problems: Vertex Cover, Independent Set, Dominating Set, Shortest Path, Matching, and Vertex Cut. For a base problem Π we write Π-D for the discovery version in the token sliding model.

Figure 1 summarizes our results. All graph classes and width parameters appearing in this introduction are defined in the preliminaries. Fellows et al. [11] and Grobler et al. [13] gave fixed-parameter tractable algorithms with respect to the parameter k for IS-D on nowhere dense graphs, for VC-D, SP-D, Mat-D, and VCut-D on general graphs and for DS-D on biclique-free graphs.

We show that IS-D, VC-D, DS-D, and Mat-D parameterized by k admit polynomial size kernels (on the aforementioned classes), while VCut-D does not admit kernels of size polynomial in k. For SP-D, we show that the problem does not admit a kernel of polynomial size parameterized by $k + b$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

As NP-hardness provides strong evidence that a problem admits no polynomial-time algorithm, $\mathsf{W}[t]$-hardness (for a positive integer t) with respect to a parameter p provides strong evidence that a problem admits no fixed-parameter tractable algorithm with respect to p. Fellows et al. [11] proved that VC-D, IS-D, and DS-D are W[1]-hard with respect to parameter b on d-degenerate graphs but provided fixed-parameter tractable algorithms on nowhere dense graphs. They also showed that these problems are slicewise polynomial (XP) with respect to the parameter treewidth and left open the parameterized complexity of these problems with respect to the parameter treewidth alone. We show that these problems remain XNLP-hard (which implies $\mathsf{W}[t]$-hardness for every positive integer t) for parameter pathwidth (even if given a path decomposition realising the pathwidth), which is greater than or equal to treewidth, and that they admit no polynomial kernels (even if given a path decomposition realising the pathwidth) with respect to the parameter $b + pw$, where pw is the pathwidth of the input graph, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Finally, we also consider the parameter feedback vertex set number (fvs), which is an upper bound on the treewidth of a graph, but is incomparable to pathwidth. We complement the parameterized complexity classification for the results of Fellows et al. [11] by showing that IS-D, VC-D, and DS-D are W[1]-hard for the parameter fvs.

Several interesting questions remain open. For instance, while their parameterized complexity was determined, the kernelization complexity of Col-D and ECut-D remains unsettled. Similarly, the kernelization complexity of IS-D and DS-D with respect to parameter k is unknown on d-degenerate and semi-ladder-free graphs, respectively, where

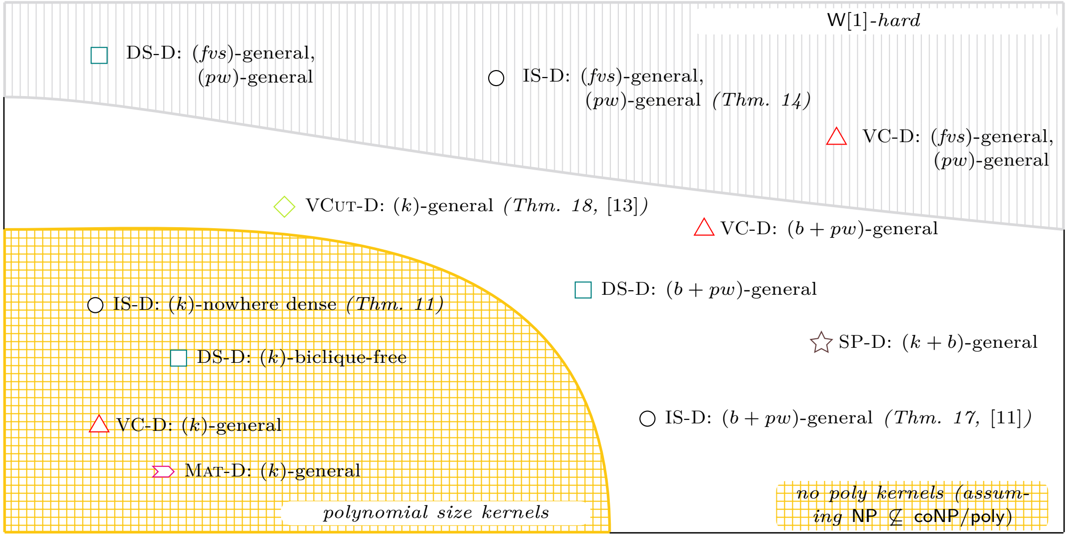

W[1]-*hard*
DS-D: (*fvs*)-general, (*pw*)-general
IS-D: (*fvs*)-general, (*pw*)-general *(Thm. 14)*
VC-D: (*fvs*)-general, (*pw*)-general
VCut-D: (*k*)-general *(Thm. 18,* [13]*)*
VC-D: ($b + pw$)-general
DS-D: ($b + pw$)-general
SP-D: ($k + b$)-general
IS-D: ($b + pw$)-general *(Thm. 17,* [11]*)*
IS-D: (*k*)-nowhere dense *(Thm. 11)*
DS-D: (*k*)-biclique-free
VC-D: (*k*)-general
Mat-D: (*k*)-general
polynomial size kernels
no poly kernels (assuming NP ⊈ coNP/poly)

Figure 1 A classification of problems into three categories: (yellow, alternatively grid) problems for which we obtain polynomial kernels, (white) those for which polynomial kernels are unlikely, and (grey, alternatively lines) those for which fixed-parameter tractable algorithms are unlikely. Each entry in a category mentions a solution discovery problem, one or more parameters (in parentheses and followed by a dash), and the graph class with respect to which the problem falls into the category. A reference in the parentheses indicates that the fixed-parameter tractability of that problem was established in the cited work. *pw* denotes the pathwidth and *fvs* denotes the feedback vertex set number of the input graph.

the problems are known to be fixed-parameter tractable. In addition, it remains open whether VCut-D parameterized by $k + b$ admits a polynomial kernel or whether Mat-D parameterized by b admits polynomial kernels on restricted classes of graphs.

1.2 Paper Outline

Due to space constraints we cannot present all results in the conference version of the paper. We have chosen to present some results for IS-D and VCut-D to give an overview of some of our techniques. All other results can be found in the full version. We collect necessary background in Section 2. We show that IS-D has a polynomial size kernel with respect to parameter k on nowhere dense classes in Section 3. Then, in Section 4 we prove XNLP-hardness with respect to pathwidth. In Section 5, we prove that polynomial kernels with respect to k are unlikely for VCut-D.

2 Preliminaries

We use the symbol $\mathbb{N}$ for the set of non-negative integers (including 0), $\mathbb{Z}$ for the set of all integers, and $\mathbb{Z}_+$ for the set of positive non-zero integers. For $k \in \mathbb{N}$, we define $[k] = \{1, \dots, k\}$ with the convention that $[0] = \varnothing$.

2.1 Graphs

We consider finite and simple graphs only. We denote the vertex set and the edge set of a graph G by $V(G)$ and $E(G)$, respectively, and denote an undirected edge between vertices u and v by uv (or equivalently vu) and a directed edge from u to v by (u, v). We use $N(v)$ to denote the set of all neighbors of v and $E(v)$ to denote the set of all edges incident with v. Furthermore, we define the closed neighborhood of v as $N[v] = N(v) \cup \{v\}$. For a set X of vertices we write $G[X]$ for the subgraph induced by X.

A sequence of edges $e_1 \dots e_\ell$ for some $\ell \geq 1$ is a (simple) path of length ℓ if every two consecutive edges in the sequence share exactly one endpoint and each other pair of edges share no endpoints. For vertices u and v, we denote the length of a shortest path $e_1 \dots e_\ell$ that connects u to v by $d(u, v)$, where $d(v, v) = 0$ for all $v \in V(G)$. For a vertex $v \in V(G)$ and a non-negative integer i, we denote by $V(v, i) = \{u \in V(G) \mid d(u, v) = i\}$.

The complete graph (clique) on n vertices is denoted by K_n and a complete bipartite graph (biclique) with parts of size m and n, respectively, by $K_{m,n}$. We present other relevant graph classes properties in what follows and refer the reader to the textbook by Diestel [7] for an in-depth review of general graph theoretic definitions.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, (X_i)_{i \in V(T)})$ where T is a tree and $X_i \subseteq V(G)$ for each $i \in V(T)$, such that

1. $\bigcup_{i \in V(T)} X_i = V(G)$,
2. for every edge $uv = e \in E(G)$, there is an $i \in V(T)$ such that $u, v \in X_i$, and
3. for every $v \in V(G)$, the subgraph T_v of T induced by $\{i \in V(T) \mid v \in X_i\}$ is connected, i. e., T_v is a tree.

We refer to the vertices of T as the *nodes* of T. For a node i, we say that the corresponding set X_i is the *bag* of i. The *width* of the tree decomposition $(T, (X_i)_{i \in V(T)})$ is $\max_{i \in V(T)} |X_i| - 1$. The *treewidth* of G, denoted $tw(G)$, is the smallest width of any tree decomposition of G. A *path decomposition* of a graph G is a tree decomposition $\mathcal{P} = (T, (X_i)_{i \in V(T)})$ in which T is a path. We represent a path decomposition $\mathcal{P}$ by the sequence of its bags only. The *pathwidth* of G, denoted $pw(G)$, is the smallest width of any path decomposition of G.

▶ **Definition 1.** *A* ***class*** $\mathcal{C}$ *of graphs has* ***bounded treewidth*** *(****bounded pathwidth****) if there exists a constant t such that all $G \in \mathcal{C}$ have treewidth (pathwidth) at most t.*

For a graph G, the *feedback vertex set number of G ($fvs(G)$)* is the minimum size of a vertex set whose deletion leaves the graph acyclic. We say a graph H is a *minor* of a graph G, denoted $H \preceq G$, if there exists a mapping that associates each vertex v of H with a non-empty connected subgraph G_v of G such that G_u and G_v are disjoint for $u \neq v$ and whenever there is an edge between u and v in H, there is an edge between a vertex of G_u and a vertex of G_v. The subgraph G_v is referred to as the *branch set* of v. We call H a *depth-r minor* of G, denoted $H \preceq_r G$, if each branch set of the mapping induces a graph of radius at most r.

▶ **Definition 2.** *A* ***class*** $\mathcal{C}$ *is* ***nowhere dense*** *if there exists a function $t : \mathbb{N} \to \mathbb{N}$ such that $K_{t(r)} \not\preceq_r G$ for all $r \in \mathbb{N}$ and all $G \in \mathcal{C}$.*

An *r-independent set* in a graph G is a set of vertices I such that the distance between any two vertices of I is at least $r + 1$. We make use of the fact that nowhere dense classes are uniform quasi-wide, as clarified by the following theorem.

▶ **Theorem 3** ([14, 17])**.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. For all $r \in \mathbb{N}$, there is a polynomial $N_r : \mathbb{N} \to \mathbb{N}$ and a constant $x_r \in \mathbb{N}$ such that following holds. Let $G \in \mathcal{C}$ and let $A \subseteq V(G)$ be a vertex subset of size at least $N_r(m)$, for a given $m \in \mathbb{N}$. Then there exists a set $X \subseteq V(G)$ of size $|X| \leq x_r$ and a set $B \subseteq A \setminus X$ of size at least m that is r-independent in $G - X$. Moreover, given G and A, such sets X and B can be computed in time $\mathcal{O}(|A| \cdot |E(G)|)$.*

A graph is said to be *d-biclique-free* it excludes the biclique $K_{d,d,}$ as a subgraph.

▶ **Definition 4.** *A* ***class*** $\mathcal{C}$ *of graphs is* ***biclique-free*** *if there exists a number d such that all $G \in \mathcal{C}$ are d-biclique-free.*

2.2 Solution Discovery

A vertex (edge) subset problem Π is a problem defined on graphs such that a solution consists of a subset of vertices (edges) satisfying certain requirements. For a vertex (edge) subset problem Π on an instance with an input graph G, a *configuration* C on G is a subset of its vertices (edges). Alternatively, a configuration can be seen as the placement of tokens on a subset of vertices (edges) in G. In the *token sliding* model, a configuration C' can be obtained (in one step) from a configuration C, written $C \vdash C'$, if $C' = (C \setminus \{y\}) \cup \{x\}$ for elements $y \in C$ and $x \notin C$ such that x and y are neighbors in G, that is, if $x, y \in V(G)$, then $xy \in E(G)$; and if $x, y \in E(G)$, then they share an endpoint. Alternatively, when a token *slides* from a vertex to an adjacent one or from an edge to an incident one, we get $C \vdash C'$. A *discovery sequence* of length ℓ in G is a sequence of configurations $C_0C_1 \dots C_\ell$ of G such that $C_i \vdash C_{i+1}$ for all $0 \leq i < \ell$.

The Π-Discovery problem is defined as follows. We are given a graph G, a configuration $S \subseteq V(G)$ (resp. $S \subseteq E(G)$) of size k (which at this point is not necessarily a solution for Π), and a budget b (a non-negative integer). We denote instances of Π-Discovery by (G, S, b). The goal is to decide whether there exists a discovery sequence $C_0C_1 \dots C_\ell$ in G for some $\ell \leq b$ such that $S = C_0$ and C_ℓ is a solution for Π. When a path decomposition is given as part of the input, the instances are denoted by $(G, \mathcal{P}_G, S, b)$ to highlight that the path decomposition $\mathcal{P}_G$ of G is provided.

2.3 Parameterized Complexity and Kernelization

Downey and Fellows [8] developed a framework for parameterized problems which include a parameter p in their input. A parameterized problem Π has inputs of the form (x, p) where $|x| = n$ and $p \in \mathbb{N}$. Fixed-parameter tractable problems belong to the complexity class FPT. The class XNLP consists of the parameterized problems that can be solved with a non-deterministic algorithm that uses $f(p) \cdot \log n$ space and $f(p) \cdot n^{\mathcal{O}(1)}$ time. The W-*hierarchy* is a collection of parameterized complexity classes $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \ldots \subseteq \mathsf{XNLP}$ where inclusions are conjectured to be strict.

For parameterized problems Π and Π', an FPT-*reduction* from Π to Π' is a reduction that given an instance (x, p) of Π produces (x', p') of Π' in time $f(p) \cdot |x|^{\mathcal{O}(1)}$ and such that $p' \leq g(p)$ where f, g are computable functions. A pl-*reduction* from Π to Π' is one that additionally computes (x', p') using $\mathcal{O}(h(p) + \log |x|)$ working space where h is a computable function. We write $\Pi \leq_{\mathsf{FPT}} \Pi'$ (resp. $\Pi \leq_{\text{pl}} \Pi'$) if there is an FPT-reduction (resp. pl-reduction) from Π to Π'. If Π is $\mathsf{W}[t]$-hard for a positive integer t and $\Pi \leq_{\mathsf{FPT}} \Pi'$, then Π' is also $\mathsf{W}[t]$-hard. If Π is XNLP-hard and $\Pi \leq_{\text{pl}} \Pi'$, then Π' is XNLP-hard and, in particular, $\mathsf{W}[t]$-hard for all $t \geq 1$.

Every problem that is in FPT admits a kernel, although it may be of exponential size or larger. Under the complexity-theoretic assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, we can rule out the existence of a polynomial kernel for certain fixed-parameter tractable problems Π. The machinery for such kernel lower bounds heavily relies on composing instances that are equivalent according to a polynomial equivalence relation [6].

▶ **Definition 5.** *An equivalence relation $\mathcal{R}$ on the set of instances of a problem Π is called a* ***polynomial equivalence relation*** *if the following two conditions hold.*

1. *There is an algorithm that given two instances x and y of Π decides whether x and y belong to the same equivalence class in time polynomial in $|x| + |y|$.*
2. *For any finite set S of instances of Π, the equivalence relation $\mathcal{R}$ partitions the elements of S into at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ classes.*

We can compose equivalent instances in more than one way. We focus here on or-cross-compositions.

▶ **Definition 6** ([3]). *Let Π' be a problem and let Π be a parameterized problem. We say that Π* ***or-cross-composes*** *into Π' if there is a polynomial equivalence relation $\mathcal{R}$ on the set of instances of Π and an algorithm that, given t instances (where $t \in \mathbb{Z}_+$) $x_1, x_2, \ldots, x_t$ belonging to the same equivalence class of $\mathcal{R}$, computes an instance (x^*, p^*) in time polynomial in $\Sigma_{i=1}^{t} |x_i|$ such that the following properties hold.*

1. *$(x^*, p^*) \in \Pi$ if and only if there exists at least one i such that x_i is a yes-instance of Π'.*
2. *p^* is bounded above by a polynomial in $\max_{i=1}^{t} |x_i| + \log t$.*

The inclusion NP $\subseteq$ coNP/poly holds if an NP-hard problem or-cross-composes into a parameterized problem Π having a polynomial kernel. As this inclusion is believed to be false, we will constantly make use of the following theorem to show that the existence of a polynomial kernel is unlikely.

▶ **Theorem 7** ([3]). *If a problem Π' is NP-hard and Π' or-cross-composes into the parameterized problem Π, then there is no polynomial kernel for Π unless NP $\subseteq$ coNP/poly.*

We refer the reader to textbooks [6, 9] for more on parameterized complexity and kernelization.

3 IS-D on Nowhere Dense Classes

Fellows et al. [11] showed that IS-D is in FPT with respect to parameters k and b on nowhere dense classes of graphs. We show in this section that IS-D has a polynomial kernel with respect to parameter k on the same.

▶ **Definition 8.** *For any instance $\mathcal{I} = (G, S, b)$ of a Π-Discovery problem for some vertex (resp. edge) selection problem Π, we call a vertex $v \in V(G) \setminus S$ (resp. $e \in E(G) \setminus S$)* ***irrelevant with respect to*** *$s \in S$ if there exists a configuration C_ℓ such that $\ell \leq b$, C_ℓ is a solution for Π, and the token on s is not on v (resp. e) in C_ℓ.*

The kernelization algorithm for nowhere dense graphs uses Theorem 3, along with other structural properties of the input graph, to form a "sunflower" and find an irrelevant vertex. It then removes from the graph some of the vertices that are irrelevant with respect to every token. A *sunflower* with p *petals* and a *core* Y is a family of sets $F_1, \ldots, F_p$ such that $F_i \cap F_j = Y$ for all $i \neq j$; the sets $F_i \setminus Y$ are petals and we require none of them to be empty [10].

▶ **Lemma 9.** *Let (G, S, b) be an instance of IS-D where $|S| = k$, and let G' be the subgraph of G induced by the vertices of $\bigcup_{s \in S, i \in [3k]} V(s, i) \cup S$. Then (G', S, b) is equivalent to (G, S, b).*

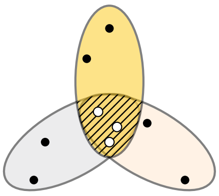


Figure 2 An example of a sunflower (with beige, yellow and grey green petals) formed by the closed neighborhoods of the vertices in $B_\mathfrak{j}$ of Theorem 11. The vertices in $B_\mathfrak{j}$ are 2-independent in $G - X$ and they have the same closed neighborhood in X (the white colored vertices).

▶ **Lemma 10.** *Let (G, S, b) be an instance of IS-D where $|S| = k$, and let $\mathcal{V} = \{v_1, v_2, \dots, v_t\}$ be a set of vertices of $G \setminus S$ such that for a given token on a vertex $s \in S$, $d(s, v_i) = d(s, v_j)$ for $i \neq j \in [t]$. If $\mathcal{A} = \{N[v_1], \dots, N[v_t]\}$ contains a sunflower with $k + 1$ petals, then any vertex whose closed neighborhood corresponds to one of those petals is irrelevant with respect to s.*

▶ **Theorem 11.** *IS-D has a polynomial kernel with respect to parameter k on nowhere dense graphs.*

Proof. Let (G, S, b) be an instance of IS-D where G is nowhere dense. Without loss of generality, we assume the graph G to be connected. For each vertex $s \in S$ and integer $i \in [3k]$, we compute $V(s, i)$. We maintain the invariant that we remove from $V(s, i)$ for each $s \in S$ and $i \in [3k]$, irrelevant vertices with respect to s (note that a vertex can appear in multiple sets $V(s, i)$).

We remove an irrelevant vertex with respect to a vertex $s \in S$ from $V(s, i)$ for an integer $i \in [3k]$ as follows. If $|V(s, i)| > N_2(2^{x_2} \cdot (k+1))$, where N_2 and x_2 are as per Theorem 3 (here $V(s, i)$ plays the role of the set A), we can compute sets $X, B \subseteq V(s, i)$ such that $|X| \leq x_2$, $|B| \geq 2^{x_2} \cdot (k + 1)$ and B is 2-independent in $G - X$. Let $\mathcal{B}' = \{B'_1, B'_2, \dots\}$ be a family of sets that partitions the vertices in B such that for any two vertices $u, v \in B$, $u, v \in B'_j$ if and only if $N[u] \cap X = N[v] \cap X$. Since $|B| \geq 2^{x_2} \cdot (k + 1)$ and $|X| \leq x_2$, at least one set $B_\mathfrak{j} \in \mathcal{B}$, for a specific $\mathfrak{j}$, contains at least $k + 1$ vertices of B. All vertices in $B_\mathfrak{j}$ have the same neighborhood in X and they are 2-independent $G - X$ (i. e., no vertex from outside of X can be in the closed neighborhood of two vertices in $B_\mathfrak{j}$); thus their closed neighborhoods form a sunflower with at least $k + 1$ petals and a core that is contained in X (Figure 2). By Lemma 10, one vertex of $B_\mathfrak{j}$ is irrelevant with respect to s and can be removed from $V(s, i)$. We can repeatedly apply Theorem 3 on the set $V(s, i)$ until $|V(s, i)| \leq N_2(2^{x_2} \cdot (k + 1))$.

We form the kernel (G', S, b) of the original instance (G, S, b) as follows. We set $V(G') = \bigcup_{s \in S, i \in [3k]} V(s, i) \cup S$. By Lemma 9, any vertex $v \in V(G)$ such $d(s, v) > 3k$ for every $s \in S$ is irrelevant with respect to every $s \in S$ and not required in the kernel (G', S, b). For each vertex $v \in V(s, i)$, for $s \in S$ and $i \in [3k]$, we add to $V(G')$ at most i vertices that are on the shortest path from s to v, if such vertices are not already present in $V(G')$. G' is the subgraph of G induced by the vertices in $V(G')$. By the end of this process, $|V(G')| \leq k + [9k^3 \cdot N_2(2^{x_2} \cdot (k + 1))]$, as for each $s \in S$ and $i \in [3k]$, $V(s, i) \leq N_2(2^{x_2} \cdot (k + 1))$ and for each vertex in the latter sets, we added to $V(G')$ at most $3k - 1$ vertices that are on a shortest path from that vertex to the vertex s. (G', S, b) is a kernel as only vertices that are irrelevant with respect to every token in S might not be in $V(G')$ and all vertices needed to move tokens from vertices in S towards an independent set using only b slides are present in $V(G')$. ◀

4 IS-D for Parameters b and Pathwidth

We now show that IS-D is XNLP-hard with respect to parameter pathwidth. By a small modification of the proof we obtain that IS-D does not have a polynomial kernel with respect to the parameter $b + pw$, where pw is the pathwidth of the input graph, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

4.1 The Minimum Maximum Outdegree Problem and Foundational Gadgets

An *orientation* of a graph H is a mapping $\lambda : E(H) \to V(H) \times V(H)$ such that $\lambda(uv) \in \{(u, v), (v, u)\}$. Given an undirected weighted graph H, a path decomposition $\mathcal{P}_H$ of H of width pw, an edge weighting $\sigma : E(H) \to \mathbb{Z}_+$ and a positive integer r (such that all integers are given in unary), the Minimum Maximum Outdegree (MMO) asks whether there exists an orientation of H such that for each $v \in V(H)$, the total weight of the edges directed away from v is at most r. We use the problem in the reductions that establish the XNLP-hardness of IS-D, VC-D, and DS-D with respect to parameter pw and the or-cross-compositions that render it unlikely for any of these problems to have a polynomial kernel with respect to parameter $b + pw$. Bodlaender et al. [1] showed that MMO is XNLP-complete with respect to pathwidth given a path decomposition realising the pathwidth.

For an instance $(H, \mathcal{P}_H, \sigma, r)$ of MMO, we define $\boldsymbol{\sigma} = \sum_{e\in E(H)} \sigma(e)$, $n = |V(H)|$ and $m = |E(H)|$. We construct for an instance $(H, \mathcal{P}_H, \sigma, r)$ of MMO, a graph G consisting of disjoint subgraphs G_e for each $e \in E(H)$ and G_v for each $v \in V(H)$. We refer to the edge-based and vertex-based subgraphs as *MMO-edge-gadgets* and *MMO-vertex-gadgets*, respectively. For an edge $e \in E(H)$ we refer to G_e as *MMO-edge-e*. Similarly, for a vertex $v \in V(H)$ we refer to G_v as *MMO-vertex-v*.

MMO-edge-e. For an edge $e = uv \in E(H)$, an MMO-edge-e G_e contains $\sigma(e) + 1$ edges with endpoints a_e^i and b_e^i for $i \in [\sigma(e) + 1]$, and an edge $e^u e^v$ such that $b_e^{\sigma(e)+1}$ is adjacent to each of e^u and e^v. We define $A_e = \cup_{i\in[\sigma(e)]}\ a_e^i$ and $B_e = \cup_{i\in[\sigma(e)]}\ b_e^i$. We refer to the connected component inside G_e (or any subdivision of G_e) containing e^u and e^v by G_e^{sel}. ⌟

MMO-vertex-v. For a vertex v in $V(H)$, an MMO-vertex-v G_v contains a *representative vertex of* v denoted by w_v, adjacent to r *target vertices of* v denoted by $x_v^1, x_v^2, \dots, x_v^r$ and one extra vertex x_v^{r+1}. Additionally, for each edge $e \in E(H)$ incident to v, the MMO-vertex-v contains $\sigma(e)$ edges with endpoints $y_e^{v(i)}$ and $z_e^{v(i)}$ for $i \in [\sigma(e)]$ such that $y_e^{v(i)}$ is adjacent to w_v, the representative vertex of v. We define $X_v = \cup_{i\in[r]}\ x_v^i$, $Y_e^v = \cup_{i\in[\sigma(e)]}\ y_e^{v(i)}$, $Z_e^v = \cup_{i\in[\sigma(e)]}\ z_e^{v(i)}$, $Y^v = \cup_{e\in E(H)}\ Y_e^v$, and $Z^v = \cup_{e\in E(H)}\ Z_e^v$. ⌟

The Graph G. We let $A = \cup_{e\in E(H)}\ A_e$, $A^+ = \cup_{e\in E(H)}\ a_e^{\sigma(e)+1}$, $B = \cup_{e\in E(H)}\ B_e$, $B^+ = \cup_{e\in E(H)}\ b_e^{\sigma(e)+1}$, $X = \cup_{v\in V(H)}\ X_v$, $X^+ = \cup_{v\in V(H)}\ x_v^{r+1}$, $Y = \cup_{v\in V(H)}\ Y^v$, and $Z = \cup_{v\in V(H)}\ Z^v$. We form G by connecting its MMO-edge-gadget vertices to its MMO-vertex-gadget vertices as follows. For a vertex $v \in V(H)$ and edge $e \in E(H)$ incident to v, we connect each vertex of B_e to a corresponding distinct vertex in Z_e^v (in other words, each b_e^i to $z_e^{v(i)}$ for $i \in [\sigma(e)]$). Similarly, we connect e^v to each vertex of Y_e^v (see Figure 3 for an example). ⌟

Our reductions must use at most $\mathcal{O}(h(pw) + \log |x|)$ working space, for an input instance of size $|x|$ and parameter pw, and a computable function h. We show that our reductions/-compositions can be performed on a *log-space transducer* and are pl-reductions. A log-space

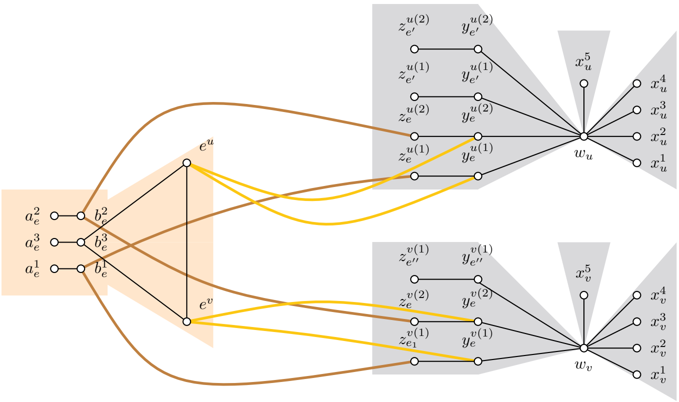


Figure 3 Edges from one MMO-edge-e, for an edge $e = uv$ for a graph H, edge weight function σ, and integer r of an MMO instance, to the MMO-vertex-u and MMO-vertex-v subgraphs in G. Brown is used for edges between vertices in B and Z and yellow is used for edges between vertices in $\{e^u, e^v\}$ and Y. $\sigma(e) = 2$ and $r = 4$.

transducer is a type of Turing machine with a read-only input tape, a read/write work tape of logarithmic size and a write-only, write-once output tape. For the graph G, we show the following lemma.

▶ **Lemma 12.** *Let $(H, \mathcal{P}_H, w, r)$ be an instance of MMO. Then, there exists a log-space transducer that transforms a path decomposition of H to one of G with width at most $pw(H)+6$. Thus, $pw(G) \leq pw(H) + 6$.*

▶ **Corollary 13.** *Given an MMO instance $(H, \mathcal{P}_H, \sigma, r)$, one can build a log-space transducer that outputs a path decomposition of G with width at most $pw(H) + 6$, along with a representation of the graph, any subset of its vertices, and an integer with at most a polynomial (in the input size) number of bits.*

4.2 Lower Bound Proofs

▶ **Theorem 14.** *IS-D is* XNLP*-hard with respect to parameter pathwidth.*

Proof. We present an fpt-reduction from MMO. Let $(H, \mathcal{P}_H, \sigma, r)$ be an instance of MMO where H is a bounded pathwidth graph, $|V(H)| = n$, $|E(H)| = m$, $\sigma : E(H) \to \mathbb{Z}_+$ such that $\sum_{e \in E(H)} \sigma(e) = \boldsymbol{\sigma}$ and $r \in \mathbb{Z}_+$ (integers are given in unary). We construct an instance $(G, \mathcal{P}_G, S, b)$ of IS-D where G is exactly as described in Section 4.1. See Figure 4. We set $S = A \cup A^+ \cup B \cup B^+ \cup Y \cup X^+$ and $b = m + 3\boldsymbol{\sigma}$. Given that all integers are given in unary, the construction of the graph G, or its path decomposition (as described in Lemma 12), and as a consequence the reduction, take time polynomial in the size of the input instance. Additionally, by Corollary 13, this reduction is a pl-reduction. We claim that $(H, \mathcal{P}_H, \sigma, r)$ is a yes-instance of MMO if and only if $(G, \mathcal{P}_G, S, b)$ is a yes-instance of IS-D.

▷ Claim 15. If $(H, \mathcal{P}_H, \sigma, r)$ is a yes-instance of MMO, then $(G, \mathcal{P}_G, S, b)$ is a yes-instance of IS-D.

Proof. Let $\lambda : E(H) \to V(H) \times V(H)$ be an orientation of the graph H such that for each $v \in V(H)$, the total weight of the edges directed out of v is at most r. In $(H, \mathcal{P}_H, \sigma, r)$, the vertices in A and B contain tokens. The same applies for the vertices in A^+ and B^+. To fix that, for each edge $e \in E(H)$ such that $\lambda(e) = (v, u)$:

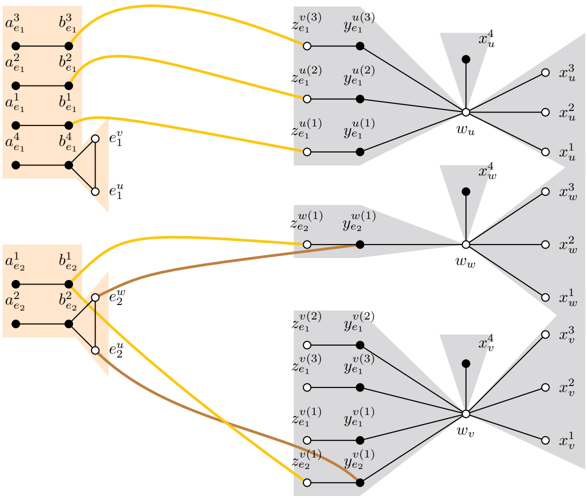


Figure 4 Parts of the graph G constructed by the reduction of Theorem 14 given an instance $(H, \mathcal{P}_H, \sigma, r)$, where H has three vertices u, v and w, and two edges $e_1 = uv$ and $e_2 = uw$, and $r = 3$. Additionally, $\sigma(e_1) = 3$ and $\sigma(e_2) = 1$. For clarity, the edges between the vertices in B_{e_1} and $Z^u_{e_1}$ are missing. The same applies for the edges between e^v_1 and the vertices of $Y^v_{e_1}$ and the edges between e^u_1 and the vertices of $Y^u_{e_1}$. Brown and yellow edges are used to highlight the different types of edges used to connect the subgraphs G_{e_1}, G_{e_2}, G_u, G_v and G_w of G, vertices in black are in S and those in white are not.

1. we slide, for each $i \in [\sigma(e)]$, the token on b^i_e to $z^{v(i)}_e$ (this consumes $\sigma(e)$ slides),
2. we move, for each $i \in [\sigma(e)]$, the token on $y^{v(i)}_e$ to any free vertex of X_v (this consumes $2\sigma(e)$ slides),
3. we slide the token on $b^{\sigma(e)+1}_e$ to e^v (this consumes 1 slide).

This constitutes $m + 3\boldsymbol{\sigma}$ slides and we get an independent set in G. Step 2 above is possible (i. e., a token-free vertex exists in X^v) since λ is an orientation of the graph H such that for each $v \in V(H)$, the total weight of the edges directed out of v is at most r. Step 3 is possible for each edge $e \in E(H)$ since in Step 2 all tokens were removed from the vertices in Y^v_e. ◁

▷ Claim 16. If $(G, \mathcal{P}_G, S, b)$ is a yes-instance of IS-D, then $(H, \mathcal{P}_H, \sigma, r)$ is a yes-instance of MMO.

Proof. The minimum number of slides used inside any induced subgraph G_e for an edge $uv = e \in E(H)$ is one and it can only be achieved by sliding the token on $b^{\sigma(e)+1}_e$ to one of either e^u or e^v. Thus, at least m slides are required inside the MMO-edge-gadgets and the budget remaining is $3\boldsymbol{\sigma}$. Additionally, each token on a vertex b^i_e in B_e, for an edge $uv = e \in E(H)$ and an integer $i \in [\sigma(e)]$ must slide to either $z^{u(i)}_e$ or $z^{v(i)}_e$, consuming $\boldsymbol{\sigma}$ slides. Since a solution that moves the token on $a^{\sigma(e)+1}_e$ but not the token on $b^{\sigma(e)+1}_e$ is not minimal, we can safely assume that the described $m + \boldsymbol{\sigma}$ slides are executed in any minimal solution.

In the same solutions, each token on a vertex $z^{u(i)}_e$ for an edge $uv = e \in E(H)$ and an integer $i \in [\sigma(e)]$ requires the token on $y^{u(i)}_e$ to slide to either e^u or w_u, utilizing as a result $\boldsymbol{\sigma}$ other slides. A token that slides from $y^{u(i)}_e$ to the vertex w_u must slide again at least once, since any independent set that is achieved through the minimal number of slides would never require the sliding of the tokens on the vertices in X^+ (the token that moves to the vertex w_u

can be moved, using one less slide, to the vertex the token on x_u^{r+1} moves to). Since G_e^{sel} can contain at most 2 tokens, a token on $y_e^{u(i)}$ that slides to the vertex e^u must either slide again at least once to a vertex, denoted $y_e^{u(i_1)}$ (for an integer $i_1 \in [\sigma(e)]$) in Y_e^u, or require another token on a vertex in G_e^{sel} to slide at least once to either a vertex, denoted $y_e^{u(i_2)}$ (for an integer $i_2 \in [\sigma(e)]$) in Y_e^u, or a vertex, denoted $y_e^{v(i_2)}$ (for an integer $i_2 \in [\sigma(e)]$) in Y_e^v, while the token initially on $y_e^{u(i)}$ stays on e^u. Given that at most $\boldsymbol{\sigma}$ slides remain in any minimal solution, and that each of the $\boldsymbol{\sigma}$ tokens initially on vertices in Y that moved to either vertices of the form ${e_1}^{u_1}$ or w_{u_1}, for an edge $e_1 \in E(H)$ incident to a vertex $u_1 \in V(H)$, uses or requires at least one additional slide, each one such token can use or require exactly one additional slide. If the token on $y_e^{u(i)}$ slides to w_u, then either in exactly one more slide it can move to a free vertex in X_u, or it can slide back to a vertex, denoted $y_{e_2}^{u(i_3)}$ (for an edge e_2 adjacent to u in H and an integer $i_3 \in [\sigma(e_1)]$) in Y^u. However, either $y_{e_2}^{u(i_3)}$ (resp. $y_e^{u(i_1)}$, $y_e^{u(i_2)}$, or $y_e^{v(i_2)}$) or its adjacent vertex, denoted $z_{e_2}^{u(i_3)}$ in Z^u (resp. $z_e^{u(i_1)}$ in Z_e^u, $z_e^{u(i_2)}$ in Z_e^u, or $z_e^{v(i_2)}$ in Z_e^v), contains a token, thus requiring at least one other additional slide, which is impossible. As a result, it can only be the case that a token on $y_e^{u(i)}$ slides to w_u and then in exactly one more slide it moves to a free vertex in X_u.

For any edge $uv = e \in E(H)$, if $e^v \in C_\ell$ (resp. $e^u \in C_\ell$), then no vertex of Y_e^v (resp. Y_e^u) appears in C_ℓ and the tokens on the vertices of Y_e^v (resp. Y_e^u) have been moved to some of the free vertices of X_v (resp. X_u). Given the latter, we produce an orientation λ to H, where $\lambda(e) = (v, u)$ (resp. $\lambda(e) = (u, v)$) if $e^v \in C_\ell$ (resp. $e^u \in C_\ell$). Since $|X_v| = |X_u| \leq r$, λ is such that the total weight directed out of any vertex $v \in V(H)$ is at most r. ◁

This concludes the proof of the theorem. ◀

We compose multiple MMO instances utilizing the construction presented in Theorem 14, and show the following.

▶ **Theorem 17.** *IS-D does not admit a polynomial kernel with respect to $b + pw$, where pw denotes the pathwidth of the input graphs, unless NP $\subseteq$ coNP/poly.*

5 VCut-D for Parameter k

Grobler et al. [13] showed that VCut-D is W[1]-hard with respect to parameter b on 2-degenerate bipartite graphs but is in FPT with respect to the parameter k on general graphs. We show that the problem admits no polynomial kernels unless NP $\subseteq$ coNP/poly. We denote an instance of VCut-D by (G, S, b, a_1, b_1) to emphasize that the solution must be a vertex cut between the vertices a_1 and b_1 in $V(G)$.

Given a graph H and an edge coloring $\phi : E(H) \to [c]$, we say ϕ is proper if, for all distinct edges $e, e_1 \in E(H)$, $\phi(e) \neq \phi(e_1)$ whenever e and e_1 share a vertex. We form our or-cross-composition from the Rainbow Matching problem, which is NP-complete even on properly colored 2-regular graphs and where every $i \in [c]$ is used exactly twice in the coloring [15]. Given a graph H, a proper edge coloring ϕ and an integer κ, the Rainbow Matching problem asks whether (H, ϕ, κ) has a a rainbow matching of size κ, i. e., a matching whose edges have distinct colors, with at least κ edges.

▶ **Theorem 18.** *There exists an or-cross-composition from Rainbow Matching into VCut-D where the parameter is the number of tokens, k. Consequently, VCut-D does not admit a polynomial kernel with respect to k, unless NP $\subseteq$ coNP/poly.*

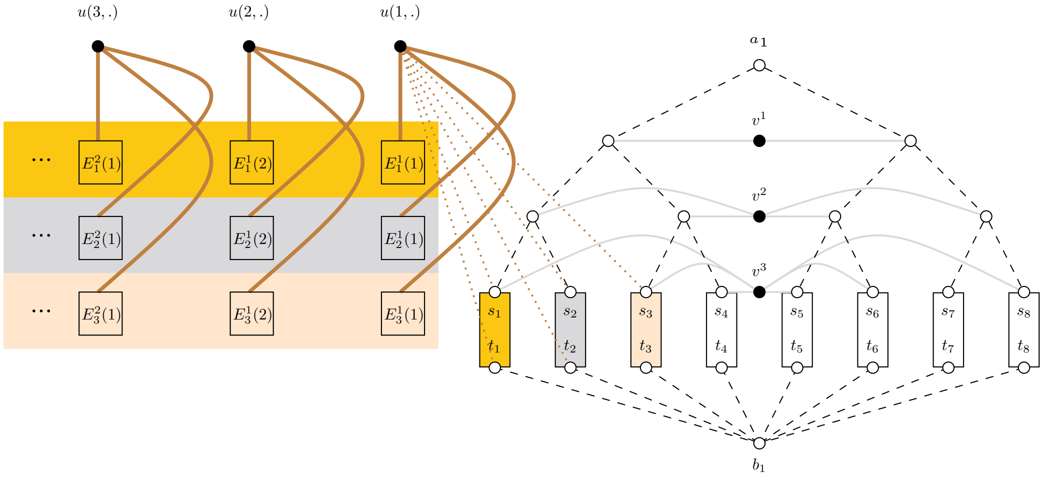


Figure 5 An illustration of the graph G formed as per the composition of Theorem 18 given input instances (H_r, ϕ_r, κ_r) for $r \in [8]$, where $\kappa_r = \kappa \geq 3$. For clarity, each graph G_r for $r \in [8]$ was replaced by a rectangle incident to two vertices s_r and t_r of G_r. Grey edges are used to illustrate how the vertices v^d for $d \in [3]$ connect to the vertices of $\mathcal{T}$. Dashed lines represent paths of length $m^3 + \log t$ between the vertices and thick edges are used to represent that a vertex is adjacent to all vertices in a set of vertices. The yellow, grey, and beige rectangular areas on the left provide a zoomed-in view of some of the content of G_1, G_2, and G_3, respectively. Particularly, they show the sets of vertices $E_1^1(1)$, $E_1^1(2)$, $E_1^2(1)$, $E_2^1(1)$, $E_2^1(2)$, $E_2^2(1)$, $E_3^1(1)$, $E_3^1(2)$, and $E_3^2(1)$. For clarity, not all (dotted line) edges between vertices of the form $u(i,j)$ for $i \in [2(\kappa-1)]$ and $j \in [m-1]$, and both vertices s_r and t_r for $r \in [8]$ are shown.

Proof. By choosing an appropriate polynomial equivalence relation $\mathcal{R}$, we may assume that we are given a family of t RAINBOW MATCHING instances (H_r, ϕ_r, κ_r), where H_r is a 2-regular graph, $|V(H_r)| = n$, $|E(H_r)| = m$, $\kappa_r = \kappa \in \mathbb{N}$, and $\phi_r : E(H_r) \to [c]$ is a mapping that properly colors H_r and in which every $i \in [c]$ is used exactly twice. We may duplicate some input instances so that $t = 2^s$ for some integer s. Note that this step at most doubles the number of input instances. The construction of the instance (G, S, b, a_1, b_1) of VCUT-D is twofold.

For each instance (H_r, ϕ_r, κ_r), we create G_r, formed of two vertices, s_r and t_r, as well as $\kappa - 1$ sets $\{E_r^1, \ldots, E_r^{\kappa-1}\}$ of $2m + 2$ vertices each. A set E_r^p for $p \in [\kappa - 1]$ contains $2m$ vertices, denoted *edge-vertices*, that represent the edges in H_r twice and two other vertices which are denoted by s_r^p and t_r^p (see Figure 6). We denote the edge-vertices in a set E_r^p as $v_{e_h}^{p,r}(1)$ ($v_{e_h}^{p,r}(2)$) to refer to the first (second) vertex representing the same edge e_h of $E(H_r)$ in E_r^p. We denote by $E_r^p(1)$ the set of all vertices $v_{e_h}^{p,r}(1)$, and by $E_r^p(2)$ the set of all vertices $v_{e_h}^{p,r}(2)$. In G_r, we connect through paths of length $m^3 + \log t$:

- s_r to each of s_r^p for $p \in [\kappa - 1]$ and t_r to each of t_r^p for $p \in [\kappa - 1]$,
- s_r^p to all vertices $v_{e_h}^{p,r}(1)$ and t_r^p to all vertices $v_{e_h}^{p,r}(2)$ for each $e_h \in E(H_r)$ and each $p \in [\kappa - 1]$,
- all vertices $v_{e_h}^{p,r}(1)$ and $v_{e_g}^{q,r}(2)$ such that $\phi_r(e_h) = \phi_r(e_g)$ for each $p \leq q \in [\kappa - 1]$,
- $v_{e_h}^{p,r}(1)$ and $v_{e_g}^{q,r}(2)$, for each $p \leq q \in [\kappa - 1]$, whenever e_h and e_g are incident in H_r,
- $v_{e_h}^{p,r}(2)$ and $v_{e_g}^{q,r}(1)$, for each $p \in [\kappa - 2]$, $q = p + 1$, whenever $e_h \neq e_g$.

We form G of all G_r for $r \in [t]$ as follows (see Figure 5). We create two global vertices a_1 and b_1 such that b_1 is connected through paths of length $m^3 + \log t$ to t_r for $r \in [t]$. Additionally, we create a binary tree $\mathcal{T}$ rooted at a_1, with $\log t + 1$ levels, and whose leaves constitute s_r for $r \in [t]$. For each depth d of $\mathcal{T}$ for $d \in \{1, \dots, \log t\}$, we create a vertex v^d that contains a token and is connected through a single edge to each vertex of $\mathcal{T}$ that is at depth d. The edges of $\mathcal{T}$ are all replaced by paths of length $m^3 + \log t$. Finally, we create $2(\kappa - 1)$ sets $\{M_1, \dots, M_{2(\kappa-1)}\}$, of $m-1$ edges each. We connect each edge $e^{(i,j)} \in M_i$ for $i \in [2(\kappa-1)]$ and $j \in [m-1]$, from one of its endpoints, denoted $u^{(i,j)}$, to each vertex $v_{e_h}^{\lceil i/2 \rceil, r}(1)$ for each $r \in [t]$ if i is odd, and to each vertex $v_{e_h}^{\lceil i/2 \rceil, r}(2)$ for each $r \in [t]$ if i is even. Additionally, we connect through paths of length $m^3 + \log t$, each s_r and t_r for $r \in [t]$ to all of $u^{(i,j)}$ for $i \in [2(\kappa - 1)]$ and $j \in [m-1]$. All vertices in the sets $\{M_1, \dots, M_{2(\kappa-1)}\}$ contain tokens. Setting $b = \log t + 2(2\kappa - 2) \cdot (m-1)$ finalizes the construction of (G, S, b, a_1, b_1). Since we perform only a polynomial number of operations per instance as well as some polynomial in t other operations while creating the tree $\mathcal{T}$ and connecting some vertices, the reduction is polynomial in $\Sigma_{i=1}^{t} |x_i|$. Additionally, k is $O(m^2 + \log t)$ since $\kappa \leq m$.

▷ Claim 19. If for some $\mathfrak{r} \in [t]$, $(H_\mathfrak{r}, \phi_\mathfrak{r}, \kappa_\mathfrak{r})$ is a yes-instance of Rainbow Matching, then the constructed instance (G, S, b, a_1, b_1) is a yes-instance of VCut-D.

Proof. Let $\mathcal{M}_\mathfrak{r}$ be a solution to the instance $(H_\mathfrak{r}, \phi_\mathfrak{r}, \kappa_\mathfrak{r})$. $\mathcal{M}_\mathfrak{r} \subseteq E(H_\mathfrak{r})$ forms a matching in $H_\mathfrak{r}$ such that $\phi_\mathfrak{r}(e_h) \neq \phi_\mathfrak{r}(e_g)$, for all $e_h, e_g \in \mathcal{M}_\mathfrak{r}$. We apply the following slides in (G, S, b, a_1, b_1) to disconnect a_1 from b_1. First, we choose one edge e_h of $\mathcal{M}_\mathfrak{r}$ and using $m-1$ slides, we slide the tokens on $u^{(1,j)}$ for $j \in [m-1]$ onto all vertices in $E_\mathfrak{r}^1(1)$ except $v_{e_h}^{1,\mathfrak{r}}(1)$. Then, using $(2\kappa - 1) \cdot (m-1)$ slides, for each $i \in [\kappa - 1]$, we choose one other edge $e_s \in \mathcal{M}_\mathfrak{r}$ and slide the tokens on $u^{(2i,j)}$ and $u^{(2i+1,j)}$ (when applicable) for $j \in [m-1]$ onto all vertices in $E_\mathfrak{r}^i(2)$ and $E_\mathfrak{r}^{i+1}(1)$ except $v_{e_s}^{i,\mathfrak{r}}(2)$ and $v_{e_s}^{i+1,\mathfrak{r}}(1)$, respectively. We slide onto $u^{(i,j)}$ for all $i \in [2(\kappa-1)]$ and $j \in [m-1]$ the tokens adjacent to the latter vertices, on the edges in $\{M_1, \dots, M_{2(\kappa-1)}\}$, using $(2\kappa - 2) \cdot (m-1)$ slides. Finally, in $\mathcal{T}$, we use the tokens on the vertices v^d for $d \in \{1, \dots, \log t\}$, to disconnect all paths from the root a_1 to all of s_r for $r \in [t] - \{\mathfrak{r}\}$, using one slide per token. This ensures that, through at most $\log t$ slides, all paths from a_1 to b_1 go through only both $s_\mathfrak{r}$ and $t_\mathfrak{r}$. Following the described steps, we have executed a total of b slides. To see that a_1 and b_1 are now disconnected, note that after the slides of the tokens on v^d for $d \in \{1, \dots, \log t\}$ are performed, all paths from a_1 to b_1 in G go through $s_\mathfrak{r}$ and $t_\mathfrak{r}$. Thus it suffices to argue that the remaining $2(2\kappa-2) \cdot (m-1)$ slides disconnect $s_\mathfrak{r}$ and $t_\mathfrak{r}$. First, if this is not the case, then no path between $s_\mathfrak{r}$ and $t_\mathfrak{r}$ goes through any $u^{(i,j)}$ for all $i \in [2(\kappa-1)]$ and $j \in [m-1]$ since the tokens that left those vertices have been replaced. Also, the last four vertices on any path between $s_\mathfrak{r}$ and $t_\mathfrak{r}$ must be $v_{e_h}^{p,\mathfrak{r}}(1)$ for some $p \in [\kappa - 1]$ and some $e_h \in E(H_\mathfrak{r})$, $v_{e_g}^{q,\mathfrak{r}}(2)$ for some $q \in \{p, \dots, \kappa - 1\}$ and some $e_g \in E(H_\mathfrak{r})$, $t_\mathfrak{r}^q$ and $t_\mathfrak{r}$. However, by construction, there exists no paths between all vertices $v_{e_h}^{p,\mathfrak{r}}(1)$ and $v_{e_g}^{q,\mathfrak{r}}(2)$ for each $p \leq q \in [\kappa - 1]$, such that $\phi_\mathfrak{r}(e_h) \neq \phi_\mathfrak{r}(e_g)$ and e_h and e_g are non-adjacent. Thus, given our choice of the free vertices remaining in $E_\mathfrak{r}^p(.)$ for all $p \in [\kappa - 1]$, no path exists between $s_\mathfrak{r}$ from $t_\mathfrak{r}$ and therefore between a_1 and b_1. ◁

▷ Claim 20. If (G, S, b, a_1, b_1) is a yes-instance of VCut-D, then there exists an integer $\mathfrak{r} \in [t]$ for which $(H_\mathfrak{r}, \phi_\mathfrak{r}, \kappa_\mathfrak{r})$ is a yes-instance of Rainbow Matching.

Proof. Assume C_ℓ for $\ell \leq b$, is a solution to (G, S, b, a_1, b_1) that is reached with only $2(2\kappa - 2) \cdot (m-1) + \log t$ slides and disconnects a_1 from b_1, then any token that slides in G slides at most once, given that everything except:

- for $d \in \{1, \dots, \log t\}$, the vertex v^d and each vertex of $\mathcal{T}$ that is at level d,
- $u^{(i,j)}$ for $i \in [2(\kappa-1)]$ and $j \in [m-1]$, to each vertex $v_{e_h}^{\lceil i/2 \rceil, r}(1)$ for each $r \in [t]$ if i is odd, and to each vertex $v_{e_h}^{\lceil i/2 \rceil, r}(2)$ for each $r \in [t]$ if i is even,
- the endpoints of each edge $e^{(i,j)} \in M_i$ for $i \in [2(\kappa-1)]$ and $j \in [m-1]$,

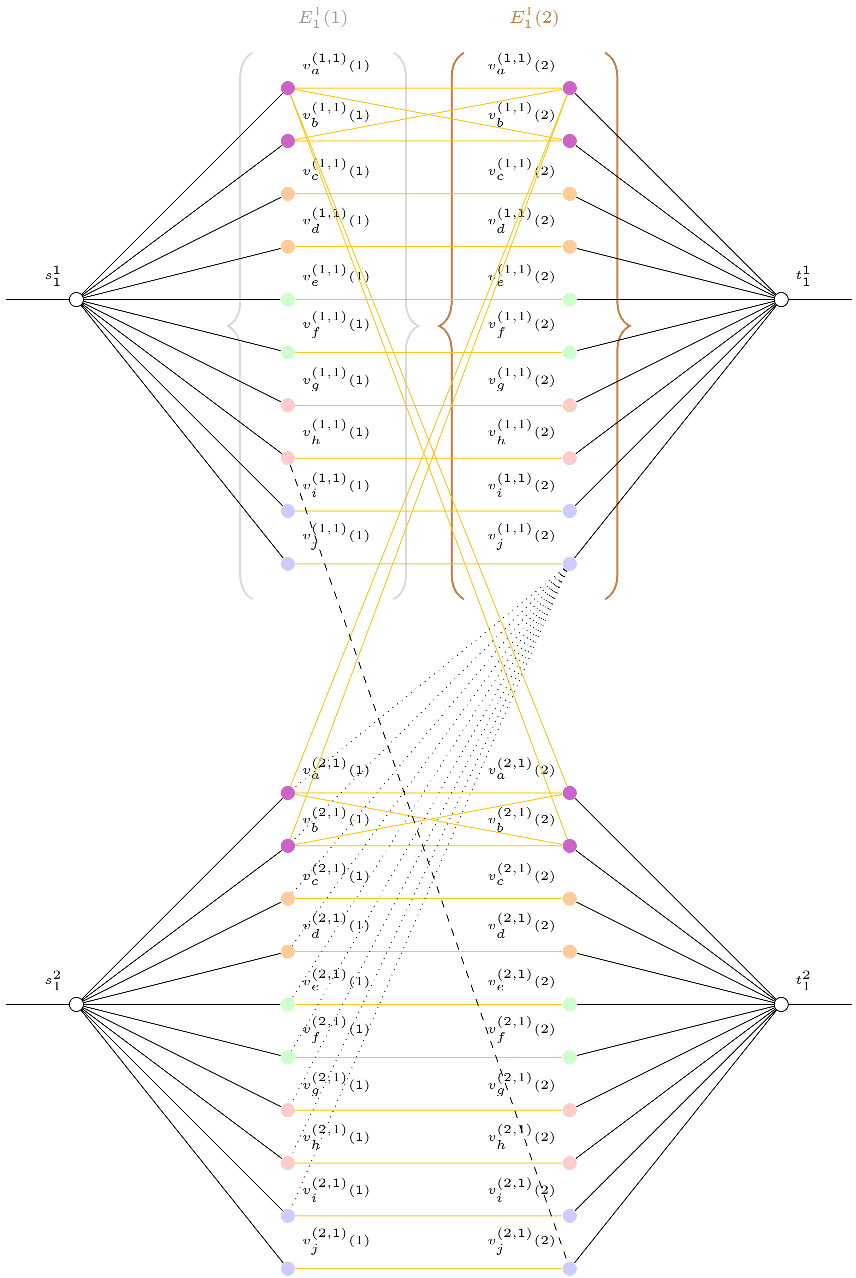


Figure 6 An illustration of E_1^1, E_1^2, s_1^1, t_1^1, s_1^2, and t_1^2 of G_1 of the or-cross-composition of Theorem 18. In H_1, the vertices are a, b, c, d, e, f, g, h, i, and j. For simplification purposes, the figure illustrates the types of edges but does not contain all edges between the illustrated vertices. Length $m^3 + \log t$ paths are represented by the edges (regular, dotted or dashed). Vertices in grey brackets are in $E_1^1(1)$ and those in brown brackets are in $E_1^1(2)$. Yellow edges are between vertices representing edges of the same color in H^1 and dotted ones between all $v_{e_h}^{p,r}(2)$ and $v_{e_g}^{q,r}(1)$ for $q = p + 1$, whenever $e_h \neq e_g$. Finally, the dashed edge shows that the edges, represented by the edge-vertices incident to it in G_1, are adjacent in H_1. In G_1, length $m^3 + \log t$ paths exist between s_1 and both of s_1^1 and s_1^2 and between t_1 and both of t_1^1 and t_1^2. No vertex in this figure contains a token (colored vertices display the colors of the edges in the instance (H_1, ϕ_1, r_1)).

is connected by paths of length $(m^3 + \log t) > b$. Thus, we know that the tokens on the vertices v^d for $d \in \{1, \ldots, \log t\}$ will have to leave some paths that go from a_1 to b_1 at least through one pair of vertices $s_\mathfrak{r}$ and $t_\mathfrak{r}$ for some $\mathfrak{r} \in [t]$ and can use at most $\log t$ slides. We know that in $G \setminus C_\ell$, no path exists between $s_\mathfrak{r}$ and $t_\mathfrak{r}$. Since no token can reach $s_\mathfrak{r}$ and $t_\mathfrak{r}$ in the allocated budget, the remaining slides can only disconnect $s_\mathfrak{r}$ from $t_\mathfrak{r}$. Note also that $u^{(i,j)} \in C_\ell$, for $i \in [2(\kappa - 1)]$ and $j \in [m-1]$ as otherwise, a path from a_1 to b_1 that goes through $s_\mathfrak{r}$, $u^{(i,j)}$ and $t_\mathfrak{r}$ will remain tokens-free. This implies that at most $m-1$ tokens can be slid into any one level $\{E^1_\mathfrak{r}(\cdot), \ldots, E^{\kappa-1}_\mathfrak{r}(\cdot)\}$. We show via an inductive argument that the set of edges in $H_\mathfrak{r}$ represented by the vertices in $\{E^1_\mathfrak{r}(\cdot), \ldots, E^{\kappa-1}_\mathfrak{r}(\cdot)\}$ but not in C_ℓ must form a matching $\mathcal{M}_\mathfrak{r}$ in $H_\mathfrak{r}$ of size $\kappa_\mathfrak{r} = \kappa$, such that for e_h, $e_g \in \mathcal{M}_\mathfrak{r}$, $\phi_\mathfrak{r}(e_h) \neq \phi_\mathfrak{r}(e_g)$ and the claim follows. Let $P(q)$ be the proposition that the set $\mathcal{E}_q$ of edges represented by vertices in $\{E^1_\mathfrak{r}(\cdot), \ldots, E^q_\mathfrak{r}(\cdot)\}$ but not in C_ℓ form a matching such that for e_h, $e_g \in \mathcal{E}_q$, $\phi_\mathfrak{r}(e_h) \neq \phi_\mathfrak{r}(e_g)$ and that vertices that remain free in $E^{q+1}_\mathfrak{r}(1)$ for $q < \kappa - 1$ represent the same edges as the vertices that remain free in $E^q_\mathfrak{r}(2)$. We show that $P(q)$ holds by induction on the levels $q = \{1, \ldots, \kappa - 1\}$.

We prove the base case by contradiction and assume that a vertex $v^{1,\mathfrak{r}}_{e_g}(2)$ that remains free in $E^1_\mathfrak{r}(2)$ either represents an edge e_g that is incident to an edge e_h represented by a vertex $v^{1,\mathfrak{r}}_{e_h}(1)$ that remains free in $E^1_\mathfrak{r}(1)$ or it holds that $\phi_\mathfrak{r}(e_g) = \phi_\mathfrak{r}(e_h)$. This implies that there exists a path between $s_\mathfrak{r}$ and $t_\mathfrak{r}$ that goes from $s_\mathfrak{r}$ to $s^1_\mathfrak{r}$, to $v^{1,\mathfrak{r}}_{e_h}(1)$, $v^{1,\mathfrak{r}}_{e_g}(2)$, $t^1_\mathfrak{r}$ and to $t_\mathfrak{r}$ and thus C_ℓ is not a solution to (G, S, b, a_1, b_1). As for the second part of the statement, assume that a vertex $v^{1,\mathfrak{r}}_{e_h}(2)$ that remains free in $E^1_\mathfrak{r}(2)$ does not represent the same edge as any of the vertices that remain free in $E^2_\mathfrak{r}(1)$, then there exists a path between $s_\mathfrak{r}$ and $t_\mathfrak{r}$ that goes through, $s^2_\mathfrak{r}$, then any of the latter vertices, followed by $v^{1,\mathfrak{r}}_{e_h}(2)$ and $t^1_\mathfrak{r}$ and thus C_ℓ is not a solution to (G, S, b, a_1, b_1). Note that the same arguments used in the base case apply for the inductive step.

In other words, given the second part of the statement, we may assume (for contradiction purposes) that a vertex $v^{i,\mathfrak{r}}_{e_g}(2)$ for $i \leq q$ (that remains free in $E^i_\mathfrak{r}(2)$) either represents an edge e_g that is incident to an edge e_h represented by a vertex $v^{i',\mathfrak{r}}_{e_h}(1)$ for $i' \leq i$ (that remains free in $E^{i'}_\mathfrak{r}(1)$) or it holds that $\phi_\mathfrak{r}(e_g) = \phi_\mathfrak{r}(e_h)$. By construction, this implies that there exists a path from $s_\mathfrak{r}$ and $t_\mathfrak{r}$ that goes from $s_\mathfrak{r}$ to $s^{i'}_\mathfrak{r}$, $v^{i',\mathfrak{r}}_{e_h}(1)$, $v^{i,\mathfrak{r}}_{e_g}(2)$, $t^i_\mathfrak{r}$, and to $t_\mathfrak{r}$ and thus C_ℓ is not a solution to (G, S, b, a_1, b_1). As for the second part of the statement, assume that a vertex $v^{q,\mathfrak{r}}_{e_h}(2)$ (that remains free in $E^q_\mathfrak{r}(2)$) does not represent the same edge as any of the vertices that remain free in $E^{q+1}_\mathfrak{r}(1)$, then there exists a path between $s_\mathfrak{r}$ and $t_\mathfrak{r}$ that goes through, $s^{q+1}_\mathfrak{r}$, then any of the latter vertices, followed by $v^{q,\mathfrak{r}}_{e_h}(2)$ and $t^q_\mathfrak{r}$ and thus C_ℓ is not a solution to (G, S, b, a_1, b_1).

Thus, $P(\kappa - 1)$ holds and the set $\mathcal{E}_{\kappa-1}$ of edges represented by vertices in $\{E^1_\mathfrak{r}(\cdot), \ldots, E^{\kappa-1}_\mathfrak{r}(\cdot)\}$ but not C_ℓ form a matching of size κ such that for e_h, $e_g \in \mathcal{E}_{\kappa-1}$, $\phi_\mathfrak{r}(e_h) \neq \phi_\mathfrak{r}(e_g)$. ◁

This concludes the proof of the theorem. ◀

References

1 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *Computing Research Repository (CoRR)*, abs/2202.06838, 2022. arXiv:2202.06838.

2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences (J. Comput. Syst. Sci.)*, 75(8):423–434, 2009. doi:10.1016/J.JCSS.2009.04.001.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics (SIAM J. Discret. Math.)*, 28(1):277–305, 2014. doi:10.1137/120880240.

4 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *Computing Research Repository (CoRR)*, abs/2204.10526, 2022. doi:10.48550/arXiv.2204.10526.

5 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic (Ann. Pure Appl. Log.)*, 84(1):119–138, 1997. doi:10.1016/S0168-0072(95)00020-8.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

10 Paul Erdös and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society (J. Lond. Math.)*, s1-35(1):85–90, 1960. doi:10.1112/jlms/s1-35.1.85.

11 Michael R. Fellows, Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Frances A. Rosamond, Daniel Schmand, and Sebastian Siebertz. On solution discovery via reconfiguration. In Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu, editors, *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 700–707. IOS Press, 2023. doi:10.3233/FAIA230334.

12 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *Journal of Computer and System Sciences (J. Comput. Syst. Sci.)*, 77(1):91–106, 2011. doi:10.1016/J.JCSS.2010.06.007.

13 Mario Grobler, Stephanie Maaz, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, and Sebastian Siebertz. Solution discovery via reconfiguration for problems in P. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 76:1–76:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.76.

14 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Transactions on Algorithms (ACM Trans. Algorithms)*, 15(2):24:1–24:19, 2019. doi:10.1145/3274652.

15 Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theoretical Computer Science (Theor. Comput. Sci.)*, 524:27–33, 2014. doi:10.1016/J.TCS.2013.12.013.

16 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

17 Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. On the number of types in sparse graphs. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 799–808. ACM, 2018. doi:10.1145/3209108.3209178.

18 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science (Theor. Comput. Sci.)*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

Approximating the Fréchet Distance When Only One Curve Is c-Packed

Joachim Gudmundsson
School of Computer Science, University of Sydney, Australia

Tiancheng Mai
School of Computer Science, University of Sydney, Australia

Sampson Wong
Department of Computer Science, University of Copenhagen, Denmark

Abstract

One approach to studying the Fréchet distance is to consider curves that satisfy realistic assumptions. By now, the most popular realistic assumption for curves is c-packedness. Existing algorithms for computing the Fréchet distance between c-packed curves require both curves to be c-packed. In this paper, we only require one of the two curves to be c-packed. Our result is a nearly-linear time algorithm that $(1+\varepsilon)$-approximates the Fréchet distance between a c-packed curve and a general curve in $\mathbb{R}^d$, for constant values of ε, d and c.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, c-packed curve, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.37

Related Version *Full Version*: `https://arxiv.org/abs/2407.05114` [15]

1 Introduction

The Fréchet distance [13] is a popular similarity measure between curves. The Fréchet distance has a variety of applications, from geographic information science [23, 24, 26] to computational biology [21, 29] and data mining [22, 28]. The Fréchet distance can be seen as the minimum leash length of a dog walking problem.

Suppose a person and a dog walk along two polygonal curves P and Q, respectively. The goal of both the person and the dog is to walk along the path, independently and at possibly different speeds, but without leaving the path or going backwards. The leash length of a given walk is defined to be the maximum distance attained between the person and the dog. The Fréchet distance is the globally minimum leash length over all possible walks.

The Fréchet distance can be computed between a pair of polygonal curves in nearly-quadratic time. Alt and Godau [1] provided an $O(n^2 \log n)$ time exact algorithm for computing the Fréchet distance. Buchin, Buchin, Meulemans and Mulzer [6] provided a randomised exact algorithm that computes the Fréchet distance in time $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on a pointer machine, and in time $O(n^2(\log\log n)^2))$ on word RAM.

Conditional lower bounds imply that the Fréchet distance problem is unlikely to admit a strongly subquadratic time algorithm. Bringmann [2] showed that, under the Strong Exponential Time Hypothesis, the Fréchet distance cannot be computed in time $O(n^{2-\delta})$ for any $\delta > 0$, if we allow for approximation factors up to 1.001. Buchin, Ophelders and Speckmann [7] showed the same conditional lower bound even if we allow for approximation factors up to 3, and even if the curves are one dimensional.

One approach to circumvent the conditional lower bounds on the Fréchet distance is to focus on curves that satisfy realistic assumptions. Realistic assumptions reflect the spatial distribution of curves from real-world data sets [17]. The most popular realistic input

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 37; pp. 37:1–37:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

assumption for curves under the Fréchet distance is c-packedness [12]. A curve $\pi \in \mathbb{R}^d$ is c-packed if for all $r > 0$, the total length of π inside any ball of radius r is upper bounded by cr.

Driemel, Har-Peled and Wenk [12] introduced the c-packedness assumption and presented a $(1 + \varepsilon)$-approximation algorithm for the Fréchet distance between a pair of c-packed curves. Their algorithm runs in $O(cn/\varepsilon + cn \log n)$ time for curves in $\mathbb{R}^d$. Bringmann and Künnemann [3] improved the running time of the algorithm to $O(\frac{cn}{\sqrt{\varepsilon}} \log^2(1/\varepsilon) + cn \log n)$ for curves in $\mathbb{R}^d$. Assuming the Strong Exponential Time Hypothesis, Bringmann [2] showed that (i) for sufficiently small constants $\varepsilon > 0$ there is no $(1 + \varepsilon)$-approximation in time $O((cn)^{1-\delta})$ for any $\delta > 0$, and (ii) in any dimension $d \geq 5$ there is no $(1 + \varepsilon)$-approximation in time $O((cn/\sqrt{\varepsilon})^{1-\delta})$ for any $\delta > 0$.

Existing algorithms [3, 12] for computing the Fréchet distance between c-packed curves require that both curves are c-packed. An open problem is whether the Fréchet distance can be approximated efficiently when only one curve is c-packed. This asymmetric case may occur if the two curves come from two different data sets. For example, in error detection we may want to match a curve containing errors to a curve close to the ground truth.

▶ **Problem 1.** *Can the Fréchet distance be approximated efficiently if only one of the two curves is c-packed? In particular, for constant values of ε, d and c, can we obtain a subquadratic time $(1+\varepsilon)$-approximation of the Fréchet distance between a c-packed curve and a general curve in $\mathbb{R}^d$?*

We resolve Problem 1 in the affirmative. Our result is an $O(c^3(n + m) \log^{2d+1}(n) \log m)$ time algorithm that $(1 + \varepsilon)$-approximates the Fréchet distance between a c-packed curve with n vertices in $\mathbb{R}^d$ and a general curve with m vertices in $\mathbb{R}^d$, where ε is a constant. In other words, to $(1 + \varepsilon)$-approximate the Fréchet distance in nearly-linear time, our result implies that it suffices to assume that only one of the two curves is c-packed. Our result is stated formally in Theorem 11. Note that for constant values of d, the running time is also polynomial in c and ε.

1.1 Related work

By now, the most popular realistic input assumption for curves under the Fréchet distance is c-packedness. The c-packedness assumption has been applied to a wide variety of Fréchet distance problems. Typically, these algorithms incur an approximation factor of $(1 + \varepsilon)$, and have a polynomial dependence on ε^{-1}. Chen, Driemel, Guibas, Nguyen and Wenk [8] study the map matching problem between a c-packed curve and realistic graph, that is, to compute a path in the graph that is most similar to the c-packed curve. Har-Peled and Raichel [19] compute the mean curve of a set of c-packed curves. The mean curve is a curve that minimises its maximum weak Fréchet distance to the set of curves. Driemel and Har-Peled [11] consider a variant of the Fréchet distance on c-packed curves, where any subcurve of the c-packed curve can be replaced by a shortcut segment. Brüning, Conradi and Driemel [4] and Gudmundsson, Huang, van Renssen and Wong [14] study two distinct variants of the subtrajectory clustering problem on c-packed curves, that is, to detect trajectory patterns by computing clusters of subcurves. Van der Hoog, Rotenberg and Wong [27] study data structures for c-packed curves under the discrete Fréchet distance. Conradi, Driemel and Kolbe [9] consider the approximate nearest neighbour problem for c-packed curves in doubling metrics. Conradi, Driemel and Kolbe [10] compute the Fréchet distance between c-packed piecewise continuous smooth curves.

Given a polygonal curve, the problem of computing its packedness value c has been considered. Gudmundsson, Sha and Wong [17] provide a 6.001-approximation algorithm that runs in $O(n^{4/3}\log^9 n)$ time for curves in $\mathbb{R}^2$. They also provided an implementation for a 2-approximation algorithm that runs in $O(n^2)$, and verified that $c < 50$ for a majority of data sets that were tested. Har-Peled and Zhou [20] provide a randomised 288.001-approximation algorithm that runs in $O(n\log^2 n)$ time and succeeds with high probability.

The c-packedness assumption can be applied to any set of edges, as a result, c-packed graphs have also been studied. Gudmundsson and Smid [18] study the map matching problem between a curve with long edges and a c-packed graph with long edges. They consider the data structure variant, where the graph is known in preprocessing time and the curve is only known at query time. Gudmundsson, Seybold and Wong [16] generalise the result of [18] and provide a map matching data structure for any c-packed graph and for any query curve.

1.2 Notation

Let $\varepsilon > 0$ be a positive real number. Without loss of generality, we can assume $0 < \varepsilon < \frac{1}{2}$, as providing a $(1+\varepsilon)$-approximation for smaller values of ε also provides a $(1+\varepsilon)$-approximation for larger values of ε.

Let d be a fixed positive integer, and let $\mathbb{R}^d$ be d-dimensional Euclidean space. A polygonal curve $P = p_1 \dots p_n$ in $\mathbb{R}^d$ consists of n vertices $\{p_i\}_{i=1}^n$ connected by $n-1$ straight line segments $\{p_ip_{i+1}\}_{i=1}^{n-1}$, where $p_i \in \mathbb{R}^d$ and $p_ip_{i+1} \subset \mathbb{R}^d$.

We define c-packedness. Let c be a positive real number. A polygonal curve P in $\mathbb{R}^d$ is c-packed if, for any radius $r > 0$ and for any ball $B(p,r)$ centred at $p \in \mathbb{R}^d$ with radius r, the set of segments in $P \cap B(p,r)$ has total length upper bounded by cr.

Next, we define the Fréchet distance. Let $P = p_1 \dots p_n$. With slight abuse of notation, define the function $P : [1,n] \to \mathbb{R}^d$ so that $P(i) = p_i$ for all integers $i \in \{1,\dots,n\}$, and $P(i+x) = (1-x)p_i + xp_{i+1}$ for all reals $x \in [0,1]$. Let $\Gamma(n)$ be the space of all continuous, non-decreasing, surjective functions from $[0,1] \to [1,n]$. For a pair of polygonal curves $P = p_1,\dots,p_n$ and $Q = q_1,\dots,q_m$, we define the Fréchet distance to be

$$d_F(P,Q) = \inf_{\substack{\alpha\in\Gamma(n)\\ \beta\in\Gamma(m)}} \max_{\mu\in[0,1]} d(P(\alpha(\mu)), Q(\beta(\mu)))$$

where $d(\cdot,\cdot)$ denotes the Euclidean distance in $\mathbb{R}^d$.

2 Decision algorithm

In this section, we solve the decision version of the Fréchet distance problem. We defer the optimisation version of the Fréchet distance problem to Section 3. Both the decision and optimisation versions will incur an approximation factor of $(1+\varepsilon)$.

We formally define the decision version. First we define the exact decision version. Let r be a positive real number. Let $P = p_1p_2\dots p_n$ be a c-packed curve in $\mathbb{R}^d$ and let $Q = q_1q_2\dots q_m$ be a general curve in $\mathbb{R}^d$. Given P, Q and r, the exact decision problem is to answer whether (i) $d_F(P,Q) \le r$ or (ii) $d_F(P,Q) > r$, where $d_F(\cdot,\cdot)$ denotes the Fréchet distance. Unfortunately, in our case, we will not be able to decide between (i) and (ii) exactly. Therefore, we instead solve the approximate decision version. In the approximate decision problem, we are additionally allowed a third option, that is (iii) to provide a $(1+\varepsilon)$-approximation for $d_F(P,Q)$.

We will build a decider for the approximate decision version, for any fixed $0 < \varepsilon < \frac{1}{2}$. Given any P, Q and r, the decider returns either (i), (ii) or (iii). The decider requires the c-packed curve to be simplified. We will first describe the simplification procedure (Section 2.1), then we will construct the fuzzy decider (Section 2.2), and finally we will combine two fuzzy deciders into a complete approximate decider (Section 2.3).

2.1 Simplification

The first step in the decision algorithm is to simplify the c-packed curve P. We will use the simplification algorithm in Driemel, Har-Peled and Wenk [12].

▶ **Fact 2** ([12]). *Given $\mu > 0$ and a polygonal curve $\pi = p_1p_2p_3...p_k$ in $\mathbb{R}^d$, we can compute in $O(k)$ time a simplification* $\text{simpl}(\pi, \mu)$ *with the following properties:*

a) *for any vertex $p \in \pi$ there exists a vertex $q \in \text{simpl}(\pi, \mu)$ such that $d(p, q) \leq \mu$,*

b) *$d_F(\pi, \text{simpl}(\pi, \mu)) \leq \mu$,*

c) *all segments in $\text{simpl}(\pi, \mu)$ have length at least μ (except the last),*

d) *if π is c-packed, then $\text{simpl}(\pi, \mu)$ is $6c$-packed.*

Proof. We state Algorithm 2.1 from [12], since we will use it in Section 3.1 to determine the critical values of our algorithm. Mark the initial vertex p_1 and set it as the current vertex. Scan the polygonal curve from the current vertex until it reaches the first vertex p_i that is at least μ away from the current vertex. Mark p_i and set it as the current vertex. Repeat this until the final vertex, and mark the final vertex. Set the marked vertices to be the simplified curve, and denote it as $\text{simpl}(\pi, \mu)$. See Figure 1. Fact 2a follows from Algorithm 2.1 in [12]. Facts 2b, 2c and 2d follow from Lemma 2.3, Remark 2.2 and Lemma 4.3 in [12]. ◀

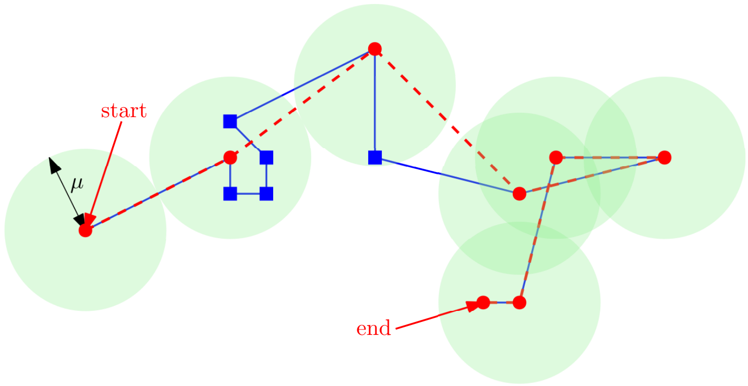


Figure 1 A polygonal trajectory P (blue) and its μ-simplification (red dashed). The vertices marked with blue squares are on P but not included in the simplification.

2.2 Fuzzy decider

The second step in the decision algorithm is to construct a fuzzy decider. Let $\varepsilon' = \varepsilon/30$. Given P, Q and r, the fuzzy decision problem is to answer whether (i) $d_F(P, Q) \leq (1+\varepsilon'/2)r$, or (ii) $d_F(P, Q) > (1 - 2\varepsilon')r$. We call the decision problem fuzzy as there is a fuzzy region $((1 - 2\varepsilon')\,r, (1 + \varepsilon'/2)\,r]$ where it would be acceptable to return either (i) or (ii). Note that unlike the complete approximate decider, for the fuzzy decider, there is no option (iii).

The overall approach in the fuzzy decider is to approximate the optimal walks along K and Q, where K is the simplification of P from Fact 2. In particular, our approach is to guess how far along K we are when we reach vertex q_i on Q. We use a layered directed graph to model the walk along K, where each layer corresponds to the walk reaching q_i on Q.

The fuzzy decision algorithm constructs a layered directed graph and searches it for a suitable walk. We divide the fuzzy decision algorithm into three steps. The first step is to query a range searching data structure [25] to construct the vertices of the graph (Section 2.2.1). The second step is to query an approximate Fréchet distance data structure [11] to construct the directed edges of the graph (Section 2.2.2). The third step is to run a breadth first search and then to return either (i) or (ii) (Section 2.2.3).

2.2.1 Constructing the vertices

The first step in the fuzzy decider is to construct the vertices of the layered directed graph. Let $\delta = \varepsilon'/2 = \varepsilon/60$. Construct the simplification $K = \mathrm{simpl}(P, \delta r)$ using Fact 2. Recall that layer i corresponds to candidate positions on K when we reach q_i on Q. Formally, define layer i to be $W_i = \{w_{i,j}\}$. All points $w_{i,j} \in W_i$ satisfy $w_{i,j} \in K$ and $d(w_{i,j}, q_i) \le 2r$. Note that $w_{i,j}$ is not necessarily a vertex of P, but rather a point on an edge of $K = \mathrm{simpl}(P, \delta r)$. To construct W_i, we require the data structure of Schwarzkopf and Vleugels [25], which is a range searching data structure for low density environments.

▶ **Definition 3.** *A set of objects Σ is k-low density if, for every box H, there are at most k objects in H that intersect it and are larger than it. The size of an object is the size of its smallest enclosing box.*

▶ **Fact 4** (Theorem 3 in [25]). *A k-low density environment Σ of n objects in $\mathbb{R}^d$ can be stored in a data structure of size $O(n \log^{d-1} n + kn)$, such that it takes $O(\log^{d-1} n + k)$ time to report all $E \in \Sigma$ that contains a given query point $q \in \mathbb{R}^d$. The data structure can be computed in $O(n \log^d n + kn \log n)$ time.*

We apply Fact 4 to the curve K. In particular, we turn K into a low-environment in $\mathbb{R}^{d+1}$ by using the trough construction of Gudmundsson, Seybold and Wong [16]. The same trough construction was also used in [5].

▶ **Lemma 5.** *Let $\delta > 0$ be fixed. Let P be a c-packed curve with n vertices in $\mathbb{R}^d$. Let $K =$ $\mathrm{simpl}(P, \delta r)$. We can preprocess K into a data structure of $O(n \log^d n + c\delta^{-1} n)$ size, so that given a query point $q \in \mathbb{R}^d$, the data structure can return in $O(\log^d n + c\delta^{-1})$ time all $O(c\delta^{-1})$ edges of K that are within a distance of $2r$ from q. The preprocessing time is $O(n \log^{d+1} n + c\delta^{-1} n \log n)$.*

Proof. The curve K is $6c$-packed by Fact 2d. Next, we generalise the trough construction of Gudmundsson, Seybold and Wong [16] to $(d+1)$-dimensions. We define a trough object in $\mathbb{R}^{d+1}$ for every segment $e \in K$ by $\mathrm{trough}(e, \delta) = \{(x_1, \dots, x_d, z) : d\left((x_1, \dots, x_d), e\right) \le 4z \le 8\delta^{-1}|e|\}$, where $d(\cdot, \cdot)$ and $|\cdot|$ are measured under the Euclidean metric in $\mathbb{R}^d$. Let T be the set of all trough objects. By Lemma 23 in [17], T is an $O(c\delta^{-1})$-low-density environment. We apply the data structure from Fact 4 on the environment T.

Given a query point $q = (x_1, \dots, x_d)$, we query the data structure for all troughs that contain the $(d+1)$-dimensional point $(x_1, \dots, x_d, r/2)$. Suppose the data structure returns a set of k objects $\{\mathrm{trough}(e_i, \delta)\}_{i=1}^k$. Then $k = O(c\delta^{-1})$, since T is an $O(c\delta^{-1})$-low-density environment, and q has zero size. From the set of k troughs we extract the set of k edges $\{e_i\}_{i=1}^k$.

The running times follow from Fact 4 and from T being an $O(c\delta^{-1})$-low-density environment. It remains to prove the correctness of the query. Recall the definition of the trough that $(x_1, \ldots, x_d, r/2) \in \text{trough}(e, \delta)$ if and only if $d\left((x_1, \ldots, x_d), e\right) \leq 4 \cdot \frac{r}{2} \leq 8\delta^{-1}|e|$. In particular, $d(q, e) \leq 2r$ covers all edges in K that intersect a ball of radius $2r$ centred at q and $4\delta^{-1}|e| \geq r$ covers all edges of length at least $\delta r/4$. Since K is also (δr)-simplified, all edges of K (except for the last edge) are at least of length δr by Fact 2c. We can check the last edge of K separately. ◀

We use Lemma 5 to construct W_i for all $1 \leq i \leq m$. Recall that $Q = q_1 \ldots q_m$. Query the data structure in Lemma 5 to obtain all edges in $K = \text{simpl}(P, \delta r)$ that are within a distance of $2r$ from q_i. Let this set of edges be T_i. Note that $|T_i| = O(c\delta^{-1})$, since K is $6c$-packed and each edge in T_i has length at least δr. For each edge $e_{i,j} \in T_i$, we choose $O(\delta^{-1})$ evenly spaced points on the chord $e_{i,j} \cap B(q_i, 2r)$, so that the distance between two consecutive points on the chord is less than δr. We add these evenly spaced points to W_i for each $e_{i,j} \in T_i$, so that in total, $|W_i| = O(c\delta^{-2})$. See Figure 2.

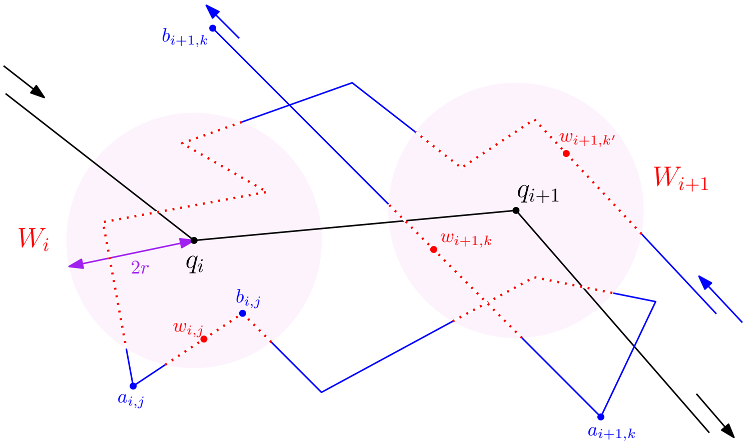


Figure 2 The general curve Q (black), the (δr)-simplification K (blue) and two candidate sets W_i and W_{i+1} (red dots). The coloured arrows indicate the order of vertices on the curve. A candidate set W_i (red dots) contains evenly spaced points on K chords that are at most a distance $2r$ away from q_i, i.e., in the violet shading. The point $w_{i,j}$ is on the edge $a_{i,j}b_{i,j}$ and the point $w_{i+1,k}$ is on the edge $a_{i+1,k}b_{i+1,k}$.

This completes the construction of W_i for $1 \leq i \leq m$. Since q_1, q_m must be matched to p_1, p_n respectively, we can simplify the sets $W_1 = \{p_1\}$ and $W_m = \{p_n\}$. The vertices of our graph are $\cup_{i=1}^{m} W_i$, which completes the first step of the construction of the fuzzy decider.

2.2.2 Constructing the edges

The second step in the fuzzy decider is to construct the edges of the layered directed graph. Each edge in the graph is a directed edge from W_i to W_{i+1} for some $1 \leq i \leq m-1$. A directed edge from $w_{i,j} \in W_i$ to $w_{i+1,k} \in W_{i+1}$ models a simultaneous walk, from $w_{i,j}$ to $w_{i+1,k}$ and from q_i to q_{i+1}, on K and Q respectively. We only add this directed edge into the graph if its associated walk is feasible. To decide whether the walk is feasible, we check two conditions. The first condition is that $w_{i,j}$ preceeds $w_{i+1,k}$ along the curve K. The second condition is whether the Fréchet distance between the subcurve $K\langle w_{i,j}, w_{i+1,k}\rangle$ and the segment $q_i q_{i+1}$ is at most r. See Figure 3. To efficiently check the second condition, we require the approximate Fréchet distance data structure of Driemel and Har-Peled [11].

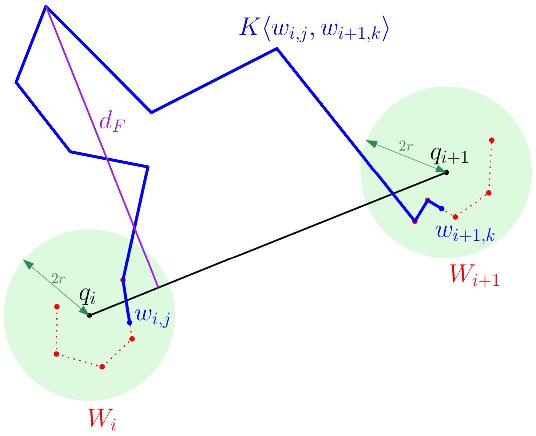


Figure 3 The Fréchet distance (purple), between a segment (q_i, q_{i+1}) (black) and subcurve $K\langle w_{i,j}, w_{i+1,k}\rangle$ (blue). A candidate set W_i (red dots) contains evenly spaced points on K chords that are at most a distance $2r$ away from q_i, i.e., in the green shading.

▶ **Fact 6** (Theorem 5.9 in [11]). *Given $\delta > 0$ and a polygonal curve Z with n vertices in $\mathbb{R}^d$, one can construct a data structure in $O(\delta^{-2d}\log^2(1/\delta)n\log^2 n)$ time that uses $O(\delta^{-2d}\log^2(1/\delta)n)$ space, such that for a query segment pq, and any two points u and v on the curve, one can $(1+\delta)$-approximate the distance $d_F(Z\langle u,v\rangle, pq)$ in $O(\delta^{-2}\log n\log\log n)$ query time.*

We construct the data structure in Fact 6 on the curve K. Let $1 \le i \le m-1$, $w_{i,j} \in W_i$ and $w_{i+1,k} \in W_{i+1}$. We query the data structure in Fact 6 to compute a $(1+\delta)$-approximation of $d_F(K\langle w_{i,j}, w_{i+1,k}\rangle, q_iq_{i+1})$. If the reported value is at most r, then we insert the directed edge from $w_{i,j}$ to $w_{i+1,k}$. We repeat this for all $1 \le i \le m-1$, $w_{i,j} \in W_i$ and $w_{i+1,k} \in W_{i+1}$. This completes the construction of the edges in the directed graph, and completes the second step of the fuzzy decider.

2.2.3 Returning either (i) or (ii)

The third step of the fuzzy decider is to run a breadth first search on the layered directed graph. Recall that $W_1 = \{p_1\}$ and $W_m = \{p_n\}$. We use the breadth first search to decide whether there is a directed path from p_1 to p_n. Recall that $\varepsilon' = \varepsilon/30$ and $\delta = \varepsilon/60$. If there is a directed path, we return (i) $d_F(P,Q) \le (1+\varepsilon'/2)r$. Otherwise, we return (ii) $d_F(P,Q) > (1-2\varepsilon')r$. Next, we prove the correctness of the fuzzy decider. We have two cases.

- There is a directed path from p_1 to p_n in the layered directed graph. Let the directed path be $c_1 \dots c_m$. Then $c_i \in W_i$ for all $1 \le i \le m$. We match the vertex q_i to c_i for all $1 \le i \le m$. We match the segment q_iq_{i+1} to the subcurve $K\langle c_i, c_{i+1}\rangle$ for all $1 \le i \le m-1$. Since there is a directed edge from c_i to c_{i+1}, we have that the estimated Fréchet distance between the segment q_iq_{i+1} and the subcurve $d_F(K\langle c_i, c_{i+1}\rangle, q_iq_{i+1})$ is at most r. Formally, we have $C_i \le r$, where

  $$d_F(K\langle c_i, c_{i+1}\rangle, q_iq_{i+1}) \le C_i \le (1+\delta)\cdot d_F(K\langle c_i, c_{i+1}\rangle, q_iq_{i+1}).$$

  In particular, we have $d_F(K\langle c_i, c_{i+1}\rangle, q_iq_{i+1}) \le r$. Taking the maximum over all $1 \le i \le m-1$, we get

  $$d_F(K,Q) \le \max_{i=1,\dots,m-1} d_F(K\langle c_i, c_{i+1}\rangle, q_iq_{i+1}) \le r.$$

By Fact 2b, we have $d_F(P, K) \leq \delta r$. Since the Fréchet distance obeys the triangle inequality, we have

$$d_F(P, Q) \leq d_F(P, K) + d_F(K, Q) \leq r + \delta r \leq (1 + \varepsilon'/2)r.$$

Therefore, it is correct to return (i) $d_F(P, Q) \leq (1 + \varepsilon'/2)r$ in the case where there is a directed path from p_1 to p_n.

- There is no directed path from p_1 to p_n in the layered directed graph. Let $r^* = d_F(P, Q)$. Suppose that for the optimal Fréchet distance between P and Q, we match $q_i \in Q$ to $p_i^* \in P$ for all $1 \leq i \leq m$. Therefore $d(q_i, p_i^*) \leq r^*$. Let $r' = d_F(K, Q)$. Suppose that for the optimal Fréchet distance between K and Q, we match $q_i \in Q$ to $k_i^* \in K$ for all $1 \leq i \leq m$. Therefore $d(q_i, k_i^*) \leq r'$. Since the Fréchet distance obeys the triangle inequality, we have $r' = d_F(K, Q) \leq d_F(P, Q) + d_F(K, P) = r^* + \delta r$.
Assume for the sake of contradiction that $r^* \leq (1 - 2\varepsilon')r$. Then, we have

$$r' \leq r^* + \delta r \leq (\delta + 1 - 2\varepsilon')\, r < r < 2r.$$

Therefore, $d(q_i, k_i^*) \leq r' < 2r$. Thus, there exists $k_i^* \in K$ that is at most a distance $2r$ away from q_i and the edge that k_i^* resides on is also at most $2r$ away from q_i. Hence, W_i is non-empty, and there exists $u_i \in W_i$ such that u_i and k_i^* share the same chord (edge) in K and $d_K(u_i, k_i^*) \leq \delta r$. In particular, there exists $u_i, v_i \in W_i$ where k_i^* is on the subcurve $K\langle u_i, v_i\rangle$, so that $d_K(u_i, k_i^*) \leq \delta r$, $d_K(v_i, k_i^*) \leq \delta r$, and $d_K(u_i, v_i) \leq \delta r$. See Figure 4.

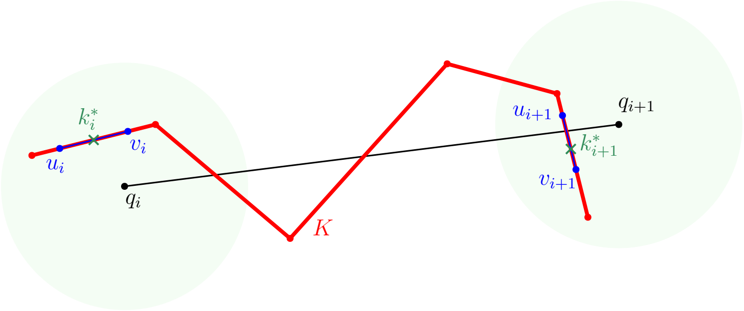


Figure 4 The point k_i^* marked with a green cross, and its immediate neighbour points u_i and v_i, marked with blue dots. Note that k_i^* is on the subcurve $K\langle u_i, v_i\rangle$, so that $d_K(u_i, k_i^*) \leq \delta r$, $d_K(v_i, k_i^*) \leq \delta r$, and $d_K(u_i, v_i) \leq \delta r$.

Consider $r_i' = d_F(K\langle u_i, u_{i+1}\rangle, q_i q_{i+1})$ when $q_i \in Q$ is matched to $u_i \in W_i \subset K$ and $q_{i+1} \in Q$ is matched to $u_{i+1} \in W_{i+1} \subset K$. Then

$$\begin{aligned} r_i' &\leq d_F(K\langle k_i^*, k_{i+1}^*\rangle, q_i q_{i+1}) + d_F(K\langle k_i^*, k_{i+1}^*\rangle, K\langle u_i, u_{i+1}\rangle) \\ &\leq r' + d_F(K\langle k_i^*, k_{i+1}^*\rangle, K\langle u_i, u_{i+1}\rangle) \\ &\leq r' + d_F(k_i^* \circ K\langle k_i^*, u_{i+1}\rangle \circ u_{i+1}k_{i+1}^*, u_i k_i^* \circ K\langle k_i^*, u_{i+1}\rangle \circ u_{i+1}) \\ &\leq r' + \max\left\{d_F(k_i^*, u_i k_i^*), d_F(K\langle k_i^*, u_{i+1}\rangle, K\langle k_i^*, u_{i+1}\rangle), d_F(u_{i+1}k_{i+1}^*, u_{i+1})\right\} \\ &\leq r' + \max\left\{d_K(k_i^*, u_i), 0, d_K(k_{i+1}^*, u_{i+1})\right\} \\ &\leq r' + \max\{\delta r, 0, \delta r\} \\ &\leq r' + \delta r \end{aligned}$$

where $\circ$ denotes the concatenation of polygonal curves. Therefore,

$$r_i' \leq r' + \delta r \leq (\delta + 1 - 2\varepsilon' + \delta)\, r = (2\delta + 1 - 2\varepsilon')\, r \leq (1 - \varepsilon')\, r.$$

Let C_i be the $(1+\delta)$-approximation of $d_F(K\langle u_i, u_{i+1}\rangle, q_i q_{i+1})$ returned by the data structure in Fact 6. Then, $r'_i \leq C_i \leq (1+\delta)r'_i$. Therefore, $C_i \leq (1+\delta)r'_i < (1+\varepsilon')r'_i \leq (1+\varepsilon')(1-\varepsilon')r < r$ for all $1 \leq i \leq m$. Hence, there is a directed edge from u_i to u_{i+1} in the layered directed graph, for all $1 \leq i \leq m-1$. In particular, $u_1 \dots u_m$ is a directed path from p_1 to p_n, which is a contradiction. We conclude that our assumption $r^* \leq (1-2\varepsilon')r$ cannot hold, and it is correct to return $r^* > (1-2\varepsilon')r$ in the case where there is no directed path from p_1 to p_n.

We obtain the following theorem.

▶ **Theorem 7** (Fuzzy decider). *Given a positive real number r, $0 < \varepsilon < \frac{1}{2}$, and a c-packed curve P with n vertices in $\mathbb{R}^d$, one can construct a data structure in $O(n\log^{d+1} n + c\varepsilon^{-1} n \log n + \varepsilon^{-2d}\log^2(1/\varepsilon) n \log^2 n)$ time that uses $O(n\log^d n + c\varepsilon^{-1}n + \varepsilon^{-2d}\log^2(1/\varepsilon)n)$ space, so that given a query curve Q with m vertices, the data structure returns in $O(\log^d n + mc^2\varepsilon^{-6}\log n \log\log n)$ query time either (i) $d_F(P,Q) \leq (1+\varepsilon'/2)r$ or (ii) $d_F(P,Q) > (1-2\varepsilon')r$.*

Proof. First, we summarise the preprocessing procedure. Let $\delta = \varepsilon/60$. We use Fact 2 to construct the simplification $K = \text{simpl}(P, \delta r)$. We use Lemma 5 to construct a range searching data structure on K, and we use Fact 6 to construct an approximate distance data structure on K. Next, we summarise the query procedure. We query Lemma 5 to construct W_i for $1 \leq i \leq m$, we query Fact 6 to construct the edges between W_i and W_{i+1} for $1 \leq i \leq m-1$, and finally we run a breadth first search. We argued correctness in Section 2.2.3. It remains to analyse preprocessing time, space, and query time.

The preprocessing time of Fact 2, Lemma 5 and Fact 6 is $O(n\log^{d+1} n + c\delta^{-1}n\log n + \delta^{-2d}\log^2(1/\delta)n\log^2 n)$. The space of the data structures in Lemma 5 and Fact 6 is $O(n\log^d n + c\delta^{-1}n + \delta^{-2d}\log^2(1/\delta)n)$. Substituting $\delta^{-1} = O(\varepsilon^{-1})$ yields the stated preprocessing time and space.

We analyse the query time. Constructing the set W_i for all $1 \leq i \leq m$ takes $O(m(\log^d n + c\delta^{-2}))$ time, since using Lemma 5 to query the set of edges close to q_i takes $O(\log^d n + c\delta^{-1})$ time, and constructing evenly spaced points takes $O(c\delta^{-2})$ time. Since $|W_i| = O(c\delta^{-2})$, the number of pairs $\cup_{i=1}^{m-1}(W_i \times W_{i+1})$ is $O(mc^2\delta^{-4})$. Querying Fact 6 to decide whether there is a directed edge takes $O(\delta^{-2}\log n \log\log n)$ time per pair. In total, constructing the edges in the layered directed graph takes $O(mc^2\delta^{-6}\log n\log\log n)$ time. Running breadth first search takes $O(mc^2\delta^{-4})$ time. The total query time is $O(\log^d n + c^2\delta^{-6}m\log n\log\log n)$. Substituting $\delta^{-1} = O(\varepsilon^{-1})$ yields the stated query time. ◀

2.3 Complete approximate decider

The third step in the decision algorithm is to use the fuzzy decider to construct a complete approximate decider. Recall that, given ε, P, Q and r, the complete approximate decider returns either (i) $d_F(P,Q) \leq r$, (ii) $d_F(P,Q) > r$, or (iii) a $(1+\varepsilon)$-approximation for $d_F(P,Q)$.

▶ **Theorem 8** (Complete approximate decider). *Given a positive real number r, $0 < \varepsilon < \frac{1}{2}$, and a c-packed curve P with n vertices in $\mathbb{R}^d$, one can construct a data structure in $O(n\log^{d+1} n + c\varepsilon^{-1}n\log n + \varepsilon^{-2d}\log^2(1/\varepsilon)n\log^2 n)$ time that uses $O(n\log^d n + c\varepsilon^{-1}n + \varepsilon^{-2d}\log^2(1/\varepsilon)n)$ space, so that given a query curve Q with m vertices in $\mathbb{R}^d$, the data structure returns in $O(\log^d n + mc^2\varepsilon^{-6}\log n\log\log n)$ query time either (i) $d_F(P,Q) \leq r$, (ii) $d_F(P,Q) > r$, or (iii) a $(1+\varepsilon)$-approximation for $d_F(P,Q)$.*

Proof. Let $r_1 = \frac{1}{1+\varepsilon'/2} r$ and $r_2 = \frac{1}{1-2\varepsilon'} r$, where $\varepsilon' = \varepsilon/30$. First, given r_1, ε, and P, construct the data structure in Theorem 7, and query the data structure on Q. Second, given r_2, ε, and P, construct the data structure in Theorem 7, and query the data structure on Q. If the first query returns (i), we return (i). If both the first and second queries return (ii), we return (ii). Otherwise, if the first query returns (ii) and the second query returns (i), we return (iii). We prove correctness in three cases.

- The first query returns (i). Then by Theorem 7, $d_F(P,Q) \leq (1+\varepsilon'/2)r_1 = r$, so returning (i) in the complete approximate decider is correct.
- Both the first and second queries return (ii). Then by Theorem 7, $d_F(P,Q) > (1-2\varepsilon')r_2 = r$, so returning (ii) in the complete approximate decider is correct.
- The first query returns (ii) and the second query returns (i). The first query implies $d_F(P,Q) > (1-2\varepsilon')\cdot r_1 = (1-2\varepsilon')\cdot\frac{1}{1+\varepsilon'/2}\cdot r$. The second query implies $d_F(P,Q) \leq (1+\varepsilon'/2)\cdot r_1 = (1+\varepsilon'/2)\cdot\frac{1}{1-2\varepsilon'}\cdot r$. Putting these together, we have

  $$d_F(P,Q) \in \left(\frac{1-2\varepsilon'}{1+\varepsilon'/2}r, \frac{1+\varepsilon'/2}{1-2\varepsilon'}r\right].$$

  Note that

  $$\frac{\frac{1+\varepsilon'/2}{1-2\varepsilon'}r}{\frac{1-2\varepsilon'}{1+\varepsilon'/2}r} = \left(\frac{1+\varepsilon'/2}{1-2\varepsilon'}\right)^2 < ((1+\varepsilon'/2)(1+4\varepsilon'))^2 < (1+6\varepsilon')^2 < 1+30\varepsilon' = 1+\varepsilon.$$

  Hence, $\frac{1-2\varepsilon'}{1+\varepsilon'/2}r$ is a $(1+\varepsilon)$-approximation of $d_F(P,Q)$, so returning (iii) in the complete approximate decider is correct.

Finally, the preprocessing time, space, and query time follow from Theorem 7. ◀

This completes the decision version of the approximate Fréchet distance problem. Next, we consider the optimisation version of the approximate Fréchet distance problem.

3 Optimisation algorithm

In Section 3.1, we apply a binary search to compute the optimal simplification. In Section 3.2, we apply parametric search to compute the Fréchet distance. In both steps, we use the complete approximate decider in Theorem 8, which incurs an approximation factor of $(1+\varepsilon)$.

3.1 Approximating the optimal simplification

First, we provide an algorithm to compute the optimal simplification of P. In particular, the optimal simplification is $K^* = \text{simpl}(P, \delta r^*)$, where $\delta = \varepsilon/60$ and $r^* = d_F(P,Q)$. Our approach is to search over the critical values of the μ-simplification algorithm in Fact 2. A critical value of the μ-simplification algorithm is a value of μ where the simplification changes. Define the set of pairwise distances of P to be $L(P) = \{d(p_i,p_j) : 1 \leq i < j \leq n\}$. We can observe that the set of pairwise distances L breaks up the positive real line into $\binom{n}{2}+1$ intervals, such that within each interval the μ-simplification does not change. This observation follows from the algorithm in Fact 2, and the same observation is made in Section 3.3.3 in [12]. Unfortunately, $|L| = O(n^2)$. To overcome this, we use approximate distance selection.

▶ **Fact 9** (Lemma 3.9 in [12]). *Given a set P of n points in $\mathbb{R}^d$, one can compute in $O(n\log n)$ time a set Z of $O(n)$ numbers, such that for any $y \in L(P)$, there exists numbers $x, x' \in Z$ such that $x \leq y \leq x' \leq 2x$.*

We can refine Fact 9 to obtain Corollary 10. We replace the 8-WSPD in Lemma 3.9 of [12] with an $8/\varepsilon$-WSPD.

▶ **Corollary 10.** *Given a set P of n points in $\mathbb{R}^d$, one can compute in $O(n/\varepsilon^d + n\log n)$ time a set Z of $O(n/\varepsilon^d)$ numbers, such that for any $y \in L(P)$, there exists numbers $x, x' \in Z$ such that $x \leq y \leq x' \leq (1+\varepsilon)x$.*

Next, we perform binary search on the set Z in Corollary 10. In particular, for $x \in Z$, we decide whether $\delta r^* < x$ or $\delta r^* > x$ by running the complete approximate decider in Theorem 8 on $r = x/\delta$, $\delta = \varepsilon/60$, P and Q. After $O(\log n)$ applications of the complete approximate decider, we obtain $\delta r^* \in [x, x']$ for a consecutive pair of elements $x, x' \in Z$. We have two cases. In the first case, we compute the optimal simplification of P, that is, $K^* = \text{simpl}(P, \delta r^*)$. In the second case, we compute a $(1+\varepsilon)$-approximation of $r^* = d_F(P, Q)$.

- If $x' > (1+\varepsilon)x$. By the contrapositive of Corollary 10, there is no $y \in L(P) \cap [x, x']$. In other words, within the interval $[x, x']$ the simplification of P does not change. Therefore, $K^* = \text{simpl}(P, x) = \text{simpl}(P, \delta r^*)$.
- If $x' \leq (1+\varepsilon)x$. Therefore, x'/δ is a $(1+\varepsilon)$-approximation of $r^* = d_F(P, Q)$, as required.

Therefore, we can compute $K^* = \text{simpl}(P, \delta r^*)$, as otherwise we would have a $(1+\varepsilon)$-approximation of r^*. The running time is dominated by the $O(\log n)$ applications of the complete approximate decider.

3.2 Approximating the Fréchet distance

From Section 3.1, we computed the simplification $K^* = \text{simpl}(P, \delta r^*)$. Let $r_1^* = \frac{1}{1+\varepsilon'/2} r^*$, $r_2^* = \frac{1}{1-2\varepsilon'} r^*$. We can use the same procedure to compute the simplifications $K_1^* = \text{simpl}(P, \delta r_1^*)$ and $K_2^* = \text{simpl}(P, \delta r_2^*)$. If $K_1^* \neq K^*$, then there must be an element $x \in Z$ in the interval $[\delta r_1^*, \delta r^*]$, so x/δ would be a $(1+\varepsilon)$-approxmation of r^*. Therefore, $K^* = K_1^*$, and similarly, $K^* = K_2^*$.

We proceed with parametric search. Note that in Section 3.1, we did not apply parametric search to compute K^* due to efficiency reasons. It is not straightforward to parallelise Fact 2, moreover, since the simplification $K^* = K_1^* = K_2^*$ does not change during the execution of the parametric search, we can avoid reconstructing the data structures in Lemma 5 and Fact 6. We obtain the following theorem.

▶ **Theorem 11.** *Given $\varepsilon > 0$, a c-packed curve P in $\mathbb{R}^d$, and a general curve Q in $\mathbb{R}^d$, one can compute a $(1+\varepsilon)$-approximation of $d_F(P, Q)$ in $O(T_s T_p \log m)$ time, where*

$$T_s = n\log^{d+1} n + c\varepsilon^{-1} n\log n + \varepsilon^{-2d}\log^2(1/\varepsilon) n\log^2 n + mc^2\varepsilon^{-6}\log n\log\log n,$$
$$T_p = \log^d n + c\varepsilon^{-1} + \varepsilon^{-2}\log n\log\log n.$$

Proof. First, we summarise the preprocessing procedure. We compute the simplification $K^* = \text{simpl}(P, \delta r^*)$ using the procedure described in Section 3.1. We build the data structures in Lemma 5 and Fact 6 on the simplified curve K^*.

Second, we summarise the query procedure. Here, we use parametric search. We use the algorithm in Theorem 8 as both the decision algorithm and the simulated algorithm. We describe the simulated algorithm. Let r be the search parameter. Let $r_1 = \frac{1}{1+\varepsilon'/2} r$ and $r_2 = \frac{1}{1-2\varepsilon'} r$. We simulate the complete approximate decider by simulating the fuzzy decider in Theorem 7 on r_1 and r_2. We divide the simulation of the fuzzy decider on r_1 into three steps. First, we compute W_i by querying the data structure in Lemma 5. We use parametric search and the decision algorithm (Theorem 8) to resolve the critical values in the query. Second, we compute the directed edges from W_i to W_{i+1} by querying the data structure in Fact 6. We apply parametric search in the same way. Third, we run a breadth first search on the layered directed graph. There are no critical values in this step, so we do not need to apply parametric search. We repeat the simulation of the fuzzy decider on r_2. Finally, by parametric search, we return the optimal value r^*.

Third, we argue correctness. If Theorem 8 returns (iii) at any point, we obtain a $(1+\varepsilon)$-approximation of r^*, and we are done. If Theorem 8 never returns (iii) at any point, we will show that the decision algorithm and the simulated algorithm are both correct. The decision algorithm is correct since we either return $r^* \leq r$ or $r^* > r$. We show the preprocessing and query procedures of the simulated algorithm are correct. In particular, we will show that we correctly simulate the execution of Theorem 8 as though $r = r^*$. The preprocessing procedure is correct, since $K^* = K_1^* = K_2^*$, so our data structures are correct for r_1^* and r_2^*. The query procedure is correct, since we can use the correct decision algorithm to resolve all critical values, and simulate the correct execution path as though $r = r^*$. Moreover, Theorem 8 (without (iii)) acts discontinuously at $r = r^*$, so r^* is a critical value of the simulated algorithm. Therefore, parametric search is able to locate r^* and return it.

Fourth, we analyse the running time. The preprocessing time is dominated by $O(\log n)$ calls to Theorem 8. The query time is dominated by parametric search. The running time of parametric search is $O(P_pT_p + T_pT_s \log P_p)$, where T_s is the sequential running time of the decision algorithm, P_p is the number of processors used in the simulated algorithm, and T_p is the number of parallel steps used by the simulated algorithm. The sequential running time is $T_s = O(n \log^{d+1} n + c\varepsilon^{-1} n \log n + \varepsilon^{-2d} \log^2(1/\varepsilon) n \log^2 n + mc^2\varepsilon^{-6} \log n \log\log n)$ by Theorem 8. The simulated algorithm can be efficiently parallelised. In particular, the simulated algorithm computes W_i by querying Lemma 5, and computes the directed edges from W_i to W_{i+1} by querying Lemma 6; these can be queried in parallel for all $1 \leq i \leq m$. Given $P_p = m$ processors, we can perform all of these queries in in $T_p = O(\log^d n + c\delta^{-1} + \delta^{-2} \log n \log\log n)$ parallel steps. The overall running time is dominated by $O(T_sT_p \log m)$, which the stated running time. ◀

We can simplify the running time if ε is constant.

▶ **Corollary 12.** *Given a constant $\varepsilon > 0$, a c-packed curve P with n vertices in $\mathbb{R}^d$, and a general curve Q with m vertices in $\mathbb{R}^d$, one can $(1+\varepsilon)$-approximate $d_F(P,Q)$ in $O(c^3(n+m)\log^{2d+1}(n)\log m)$ time.*

4 Conclusion

In this paper, we provide an $O(c^3(n+m)\log^{2d+1}(n)\log m)$ time algorithm to $(1+\varepsilon)$-approximate the Fréchet distance between two curves in $\mathbb{R}^d$, in the case when only one curve is c-packed and ε is constant. The running time is nearly-linear if c and d are also constant. An open problem is whether the running time can be improved, in particular, whether the dependence on ε, c, d, $\log n$ or $\log m$ can be reduced. Another open problem is whether we can obtain results for related problems when only one of the two curves is c-packed. Yet another open problem is whether similar results can be obtained for other realistic input curves. In particular, can the Fréchet distance be $(1+\varepsilon)$-approximated in subquadratic time when only one of the curves is κ-bounded, or when only one of the curves is ϕ-low density?

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05:75–91, 1995. doi:10.1142/s0218195995000064.

2 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 2014. doi:10.1109/focs.2014.76.

3 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27:85–119, 2017. doi:10.1142/s0218195917600056.

4 Frederik Brüning, Jacobus Conradi, and Anne Driemel. Faster approximate covering of subcurves under the Fréchet distance. In *Proceedings of the 30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.28.

5 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Aleksandr Popov, and Sampson Wong. Map-matching queries under Fréchet distance on low-density spanners. In *Proceedings of the 40th International Symposium on Computational Geometry, SoCG 2024*, volume 293 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SOCG.2024.27.

6 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

7 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2887–2901. SIAM, 2019. doi:10.1137/1.9781611975482.179.

8 Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering and Experiments, ALENEX 2011*, pages 75–83. SIAM, 2011. doi:10.1137/1.9781611972917.8.

9 Jacobus Conradi, Anne Driemel, and Benedikt Kolbe. $(1+\epsilon)$-ANN data structure for curves via subspaces of bounded doubling dimension. *Comput. Geom. Topol.*, 3(2):6:1–6:22, 2024. URL: https://www.cgt-journal.org/index.php/cgt/article/view/45.

10 Jacobus Conradi, Anne Driemel, and Benedikt Kolbe. Revisiting the Fréchet distance between piecewise smooth curves. *CoRR*, abs/2401.03339, 2024. doi:10.48550/arXiv.2401.03339.

11 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

12 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. doi:10.1007/s00454-012-9402-z.

13 M. Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22(1):1–72, 1906. doi:10.1007/bf03018603.

14 Joachim Gudmundsson, Zijin Huang, André van Renssen, and Sampson Wong. Computing a subtrajectory cluster from c-packed trajectories. In *Proceedings of the 34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.34.

15 Joachim Gudmundsson, Michael Mai, and Sampson Wong. Approximating the Fréchet distance when only one curve is c-packed. *CoRR*, abs/2407.05114, 2024. doi:10.48550/arXiv.2407.05114.

16 Joachim Gudmundsson, Martin P. Seybold, and Sampson Wong. Map matching queries on realistic input graphs under the Fréchet distance. *ACM Trans. Algorithms*, 20(2):14, 2024. doi:10.1145/3643683.

17 Joachim Gudmundsson, Yuan Sha, and Sampson Wong. Approximating the packedness of polygonal curves. *Comput. Geom.*, 108:101920, 2023. doi:10.1016/J.COMGEO.2022.101920.

18 Joachim Gudmundsson and Michiel H. M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Comput. Geom.*, 48(6):479–494, 2015. doi:10.1016/J.COMGEO.2015.02.003.

19 Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014. doi:10.1145/2532646.

20 Sariel Har-Peled and Timothy Zhou. How packed is it, really? *CoRR*, abs/2105.10776, 2021. arXiv:2105.10776.

21 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 06(01):51–64, 2008. doi:10.1142/s0219720008003278.

22 Richard J. Kenefic. Track clustering using Fréchet distance and minimum description length. *Journal of Aerospace Information Systems*, 11(8):512–524, 2014. doi:10.2514/1.i010170.

23 Patrick Laube. *Computational Movement Analysis*. Springer International Publishing, 2014. doi:10.1007/978-3-319-10268-9.

24 Peter Ranacher and Katerina Tzavella. How to compare movement? A review of physical movement similarity measures in geographic information science and beyond. *Cartography and Geographic Information Science*, 41(3):286–307, 2014. doi:10.1080/15230406.2014.890071.

25 Otfried Schwarzkopf and Jules Vleugels. Range searching in low-density environments. *Information Processing Letters*, 60(3):121–127, 1996. doi:10.1016/s0020-0190(96)00154-8.

26 Kevin Toohey and Matt Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, 2015. doi:10.1145/2782759.2782767.

27 Ivor van der Hoog, Eva Rotenberg, and Sampson Wong. Data structures for approximate discrete Fréchet distance. *CoRR*, abs/2212.07124, 2022. doi:10.48550/arXiv.2212.07124.

28 Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. An effectiveness study on trajectory similarity measures. In *Proceedings of the 24th Australasian Database Conference – Volume 137*, ADC '13, pages 13–22. Australian Computer Society, Inc., 2013.

29 Tim Wylie and Binhai Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1372–1383, 2013. doi:10.1109/tcbb.2013.17.

Basis Sequence Reconfiguration in the Union of Matroids

Tesshu Hanaka
Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

Yuni Iwamasa
Graduate School of Informatics, Kyoto University, Japan

Yasuaki Kobayashi
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Yuto Okada
Graduate School of Informatics, Nagoya University, Japan

Rin Saito
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract

Given a graph G and two spanning trees T and T' in G, Spanning Tree Reconfiguration asks whether there is a step-by-step transformation from T to T' such that all intermediates are also spanning trees of G, by exchanging an edge in T with an edge outside T at a single step. This problem is naturally related to matroid theory, which shows that there always exists such a transformation for any pair of T and T'. Motivated by this example, we study the problem of transforming a sequence of spanning trees into another sequence of spanning trees. We formulate this problem in the language of matroid theory: Given two sequences of bases of matroids, the goal is to decide whether there is a transformation between these sequences. We design a polynomial-time algorithm for this problem, even if the matroids are given as basis oracles. To complement this algorithmic result, we show that the problem of finding a shortest transformation is NP-hard to approximate within a factor of $c \log n$ for some constant $c > 0$, where n is the total size of the ground sets of the input matroids.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Combinatorial reconfiguration, Matroids, Polynomial-time algorithm, Inapproximability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.38

Funding *Tesshu Hanaka*: Supported by JSPS KAKENHI Grant Numbers JP21K17707, JP21H05852, JP22H00513, and JP23H04388.
Yuni Iwamasa: Supported by JSPS KAKENHI Grant Numbers JP22K17854, JP24K02901, JP24K21315.
Yasuaki Kobayashi: Supported by JSPS KAKENHI Grant Numbers JP20H00595, JP23K28034, JP24H00686, and JP24H00697.
Yuto Okada: Supported by JST SPRING Grant Number JPMJSP2125.
Rin Saito: Supported by JST SPRING Grant Number JPMJSP2114.

1 Introduction

In *reconfiguration problems* (see [8, 15] for introductory material), given two (feasible) configurations in a certain system, the objective is to determine whether there exists a step-by-step transformation between these configurations such that all intermediate configurations are also feasible. Among numerous reconfiguration problems studied in the literature, one of the first problems explicitly recognized as a reconfiguration problem is Spanning Tree

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 38; pp. 38:1–38:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Reconfiguration. In this problem, given two spanning trees T, T' in a (multi)graph G, one is asked to find a transformation from one spanning tree T into the other spanning tree T' by repeatedly exchanging a single edge (i.e., $T - e + f$ for an edge $e \in E(T)$ and $f \in E(G) \setminus E(T)$), such that all intermediates are also spanning trees in G. Ito et al. [9][1] observed that one can always find such a transformation with exactly $|E(T) \setminus E(T')|$ exchanges by exploiting a well-known property of *matroids*.

Let E be a finite set and let $\mathcal{B} \subseteq 2^E$ be a nonempty collection of subsets of E that satisfies the following *basis exchange axiom*: for distinct $B, B' \in \mathcal{B}$ and $x \in B \setminus B'$, there is $y \in B' \setminus B$ satisfying $B - x + y \in \mathcal{B}$. Then, the pair $M = (E, \mathcal{B})$ is called a *matroid*, and each set in $\mathcal{B}$ is called a *basis* of M. For a connected graph G with edge set $E(G)$, let $\mathcal{T}$ be the collection of all edge subsets, each of which induces a spanning tree in G. Then it is well known that $\mathcal{T}$ satisfies the basis exchange axiom: for each $e \in E(T) \setminus E(T')$, there is an edge $f \in E(T') \setminus E(T)$ such that $T - e + f$ is a spanning tree of G. Hence, the pair $(E(G), \mathcal{T})$ is a matroid, called a *graphic matroid*. This also allows us to find a transformation of $|E(T) \setminus E(T')|$ exchanges for Spanning Tree Reconfiguration. Since every transformation between T and T' requires at least $|E(T) \setminus E(T')|$ exchanges, this is a shortest one among all transformations.

In this paper, we address a natural extension of Spanning Tree Reconfiguration. Let G be a (multi)graph. We say that a sequence of k spanning trees $(T_1, \dots, T_k)$ of G is *feasible* if the spanning trees are edge-disjoint. A pair of two feasible sequences of spanning trees $\mathbb{T} = (T_1, \dots, T_k)$ and $\mathbb{T}' = (T'_1, \dots, T'_k)$ is said to be *adjacent* if there is an index $1 \leq i \leq k$ such that $T_j = T'_j$ for $1 \leq j \leq k$ with $i \neq j$ and $T'_i = T_i - e + f$ for some $e \in E(T_i)$ and $f \in E(G) \setminus E(T_i)$. Given two feasible sequences of k spanning trees $\mathbb{T} = (T_1, \dots, T_k)$ and $\mathbb{T}' = (T'_1, \dots, T'_k)$ of a graph $G = (V, E)$, Spanning Tree Sequence Reconfiguration asks whether there are feasible sequences $\mathbb{T}_0, \dots, \mathbb{T}_\ell$ such that $\mathbb{T}_0 = \mathbb{T}$, $\mathbb{T}_\ell = \mathbb{T}'$, and $\mathbb{T}_{i-1}$ and $\mathbb{T}_i$ are adjacent for all $1 \leq i \leq \ell$. This type of problem naturally extends conventional reconfiguration problems by enabling a "simultaneous transformation" of multiple mutually exclusive solutions.

To address Spanning Tree Sequence Reconfiguration, we consider a more general problem, called Basis Sequence Reconfiguration. Let $\mathbb{M} = (M_1, \dots, M_k)$ be a sequence of matroids, where $M_i = (E_i, \mathcal{B}_i)$ for $1 \leq i \leq k$. Let us note that E_i and E_j may not be disjoint for distinct i and j. A *basis sequence* of $\mathbb{M}$ is a sequence $\mathbb{B} = (B_1, \dots, B_k)$ such that B_i is a basis of M_i (i.e., $B_i \in \mathcal{B}_i$). A basis sequence $\mathbb{B} = (B_1, \dots, B_k)$ is said to be *feasible* for $\mathbb{M}$ if $B_i \cap B_j = \emptyset$ for $1 \leq i < j \leq k$. A pair of feasible basis sequences $\mathbb{B} = (B_1, \dots, B_k)$ and $\mathbb{B}' = (B'_1, \dots, B'_k)$ is said to be *adjacent* if there is an index $1 \leq i \leq k$ such that $B_j = B'_j$ for $1 \leq j \leq k$ with $i \neq j$ and $B'_i = B_i - x + y$ for some $x \in B_i$ and $y \in E_i \setminus B_i$. A feasible basis sequence $\mathbb{B}$ is *reconfigurable to* a feasible basis sequence $\mathbb{B}'$ if there are feasible basis sequences $\mathbb{B}_0, \dots, \mathbb{B}_\ell$ of $\mathbb{M}$ such that $\mathbb{B}_0 = \mathbb{B}$, $\mathbb{B}_\ell = \mathbb{B}'$, and $\mathbb{B}_{i-1}$ and $\mathbb{B}_i$ are adjacent for all $1 \leq i \leq \ell$. We refer to such a sequence $\langle \mathbb{B}_0, \dots, \mathbb{B}_\ell \rangle$ as a *reconfiguration sequence* between $\mathbb{B}$ and $\mathbb{B}'$. Our problem is formally defined as follows.

Basis Sequence Reconfiguration

Input: A tuple $\mathbb{M} = (M_1, \dots, M_k)$ of k matroids and feasible basis sequences $\mathbb{B} = (B_1, \dots, B_k)$ and $\mathbb{B}' = (B'_1, \dots, B'_k)$.

Question: Determine if $\mathbb{B}$ is reconfigurable to $\mathbb{B}'$.

[1] More specifically, they considered a weighted version of this problem.

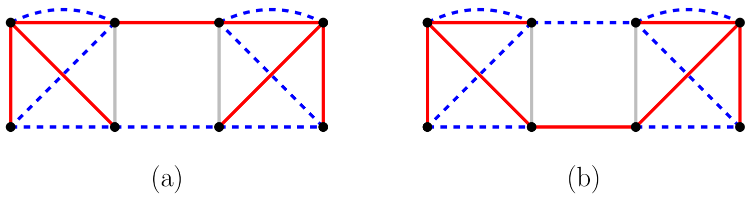


Figure 1 The figure illustrates an instance in which a pair of edge-disjoint spanning trees (a) cannot be transformed into the other pair (b), where the spanning trees are indicated by dashed blue lines and solid red lines.

Note that if $M_i = (E(G), \mathcal{T})$ for every i, $\mathbb{B} = \mathbb{T}$, and $\mathbb{B}' = \mathbb{T}'$, Basis Sequence Reconfiguration is equivalent to Spanning Tree Sequence Reconfiguration.

We also consider an optimization variant of Basis Sequence Reconfiguration: Given an instance of Basis Sequence Reconfiguration, the goal is to find a shortest reconfiguration sequence between $\mathbb{B}$ and $\mathbb{B}'$. We refer to this problem as Shortest Basis Sequence Reconfiguration.

We investigate the computational complexity of Basis Sequence Reconfiguration. In this paper, matroids are sometimes given as *basis oracles*, that is, given a set $X \subseteq E$ of a matroid $M = (E, \mathcal{B})$, the basis oracle (of M) returns true if and only if $X \in \mathcal{B}$. In such a case, we can access $\mathcal{B}$ through this oracle and assume that the basis oracle can be evaluated in polynomial in $|E|$. Our main contribution is as follows.

▶ **Theorem 1.** Basis Sequence Reconfiguration *can be solved in polynomial time, assuming that the input matroids are given as basis oracles. Moreover, if the answer is affirmative, we can compute a reconfiguration sequence between given two feasible basis sequences in polynomial time as well.*

This result nontrivially generalizes the previous result of [9]. It would be worth mentioning that, in contrast to Spanning Tree Reconfiguration, our problem Spanning Tree Sequence Reconfiguration has infinitely many no-instances (see Figure 1 for an example).

A natural extension of Basis Sequence Reconfiguration is to find a *shortest* reconfiguration sequence. Unfortunately, we show that it is hard to find it in polynomial time, even for approximately shortest reconfiguration sequences.

▶ **Theorem 2.** Shortest Basis Sequence Reconfiguration *is* NP*-hard even if the input sequence* $\mathbb{M}$ *consists of two partition matroids. Furthermore, unless* P = NP, Shortest Basis Sequence Reconfiguration *cannot be approximated in polynomial time within a factor of* $c \log n$ *for some constant* $c > 0$, *where* n *is the total size of the ground sets of the input matroids.*

Related work

Due to the property of "one-by-one exchange" in combinatorial reconfiguration, various reconfiguration problems are naturally related to matroids [1, 5, 9, 11, 12, 13]. As mentioned above, Ito et al. [9] studied Spanning Tree Reconfiguration and showed that every spanning tree can be transformed into any other spanning tree in a graph. Given this fact, Ito et al. [12] further considered a directed analogue of this problem, in which the objective is to determine whether two arborescences (i.e., directed spanning trees) in a directed graph are transformed into each other. Contrary to the undirected counterpart, for a (weakly) connected directed graph $D = (V, A)$, the pair $(A, \mathcal{F})$ is not a matroid in general, where $\mathcal{F}$ denotes the family of arc sets $F \subseteq A$, each of which forms an arborescence of D, while it is the collection of common bases of two matroids, i.e., $\mathcal{F} = \mathcal{B}_1 \cap \mathcal{B}_2$ for some matroids $(A, \mathcal{B}_1)$ and $(A, \mathcal{B}_2)$. They still showed that every arborescence can be transformed into any other arborescence in a directed graph. As a generalization of [12], Kobayashi, Mahara, and Schwarcz [13] studied the reconfiguration problem of (not the *sequence* of but) the *union* of disjoint arborescences. Namely, in their setting, a feasible solution is the union $\bigcup_{i=1}^{k} F_i$ of disjoint arborescences $F_1, F_2, \ldots, F_k$, and two feasible solutions $\bigcup_{i=1}^{k} F_i$ and $\bigcup_{i=1}^{k} F_i'$ are adjacent if and only if there are $x \in \bigcup_{i=1}^{k} F_i \setminus \bigcup_{i=1}^{k} F_i'$ and $y \in \bigcup_{i=1}^{k} F_i' \setminus \bigcup_{i=1}^{k} F_i$ such that $\bigcup_{i=1}^{k} F_i - x + y = \bigcup_{i=1}^{k} F_i'$. We note that even if two feasible solutions $\bigcup_{i=1}^{k} F_i$ and $\bigcup_{i=1}^{k} F_i'$ are adjacent in the sense of [13], the corresponding tuples $(F_1, F_2, \ldots, F_k)$ and $(F_1', F_2', \ldots, F_k')$ may not be adjacent in our sense. It is worth mentioning that the reconfiguration problem of the *union* of disjoint bases is trivially solvable, since it is just the reconfiguration problem of bases of the union of matroids; see Section 2 for the definition of the matroid union. For other reconfiguration problems related to (common bases of) matroids, see [5, 12].

Our work is highly related to a recent work of Bérczi, Mátravölgyi, and Schwarcz [1]. They considered the symmetric exchange version of our problem, where two (not necessarily feasible) basis sequences $\mathbb{B} = (B_1, \ldots, B_k)$ and $\mathbb{B}' = (B_1', \ldots, B_k')$ are adjacent if there are $x \in B_i \setminus B_j$ and $y \in B_j \setminus B_i$ such that

$$\mathbb{B}' = (B_1, \ldots, B_{i-1}, B_i - x + y, B_{i+1}, \ldots, B_{j-1}, B_j - y + x, B_{j+1}, \ldots, B_k).$$

This reconfiguration problem has received considerable attention as its reconfigurability is essentially equivalent to White's conjecture [20]. (See [1] for a comprehensive overview of White's conjecture.) In particular, the conjecture states that for any pair of two feasible basis sequences $\mathbb{B} = (B_1, \ldots, B_k)$ and $\mathbb{B}' = (B_1', \ldots, B_k')$, $\mathbb{B}$ is reconfigurable to $\mathbb{B}'$ (by symmetric exchanges) if and only if $\bigcup_{i=1}^{k} B_i = \bigcup_{i=1}^{k} B_i'$. The conjecture is confirmed for graphic matroids [2, 6], which means that for every pair of sequences of edge-disjoint k spanning trees $(T_1, \ldots, T_k)$ and $(T_1', \ldots, T_k')$ in a graph, one is reconfigurable to the other by symmetric exchanges if $\bigcup_{i=1}^{k} E(T_i) = \bigcup_{i=1}^{k} E(T_i')$. This is in contrast to our setting, having an impossible case as seen in Figure 1.

We would like to emphasize that our setting is also quite natural as it can be seen as a reconfiguration problem in the *token jumping model*, which is best studied in the context of combinatorial reconfiguration [8, 15]. In particular, our problem can be regarded as a reconfiguration problem for *multiple* solutions. One of the most well-studied problems in this context is Coloring Reconfiguration [3, 4, 7], which can be seen as a multiple solution variant of Independent Set Reconfiguration. There are several results working on reconfiguration problems for multiple solutions, such as Disjoint Paths Reconfiguration [10] and Disjoint Shortest Paths Reconfiguration [18].

2 Preliminaries

For a positive integer n, let $[n] := \{1, 2, \ldots, n\}$. For integers p and q with $p \leq q$, let $[p, q] := \{p, p+1, \ldots, q-1, q\}$. For sets X and Y, the *symmetric difference* of X and Y is defined as $X \triangle Y := (X \setminus Y) \cup (Y \setminus X)$.

Let E be a finite set and let $\mathcal{B} \subseteq 2^E$ be a nonempty collection of subsets of E. We say that $M = (E, \mathcal{B})$ is a *matroid* if for $B, B' \in \mathcal{B}$ and $x \in B \setminus B'$, there is $y \in B' \setminus B$ satisfying $(B \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. For notational convenience, we may write $B - x + y$ instead of $(B \setminus \{x\}) \cup \{y\}$. Each set in $\mathcal{B}$ is called a *basis* of M. It is easy to verify that each basis of

M has the same cardinality, which is called the *rank* of M. In this paper, we may assume that, unless explicitly stated otherwise, matroids are given as *basis oracles*. In this model, we can access a matroid $M = (E, \mathcal{B})$ through an oracle that decides whether $X \in \mathcal{B}$ for given $X \subseteq E$.[2] We also assume that we can evaluate this query in time $|E|^{O(1)}$.

Let $M_1 = (E_1, \mathcal{B}_1), \ldots, M_k = (E_k, \mathcal{B}_k)$ be k matroids and let $\mathbb{M} = (M_1, \ldots, M_k)$. For $i \in [k]$, let B_i be a basis of M_i. A tuple $\mathbb{B} = (B_1, \ldots, B_k)$ of bases is called a *basis sequence* of $\mathbb{M}$. Since E_i and E_j may have an intersection for distinct i and j, B_i and B_j are not necessarily disjoint. We say that $\mathbb{B}$ is *feasible* if $B_i \cap B_j = \emptyset$ for distinct $i, j \in [k]$. For two feasible basis sequences $\mathbb{B} = (B_1, \ldots, B_k)$ and $\mathbb{B}' = (B'_1, \ldots, B'_k)$ of $\mathbb{M}$, we say that $\mathbb{B}$ is *adjacent* to $\mathbb{B}'$ if there is an index $i \in [k]$ such that $B_j = B'_j$ for $j \in [k] \setminus \{i\}$ and $B'_i = B_i - x + y$ for some $x \in B_i$ and $y \in E_i \setminus B_i$. A *reconfiguration sequence* between $\mathbb{B}$ and $\mathbb{B}'$ is a tuple of feasible basis sequences $\langle \mathbb{B}_0, \mathbb{B}_1, \ldots, \mathbb{B}_\ell \rangle$ such that $\mathbb{B}_0 = \mathbb{B}$, $\mathbb{B}_\ell = \mathbb{B}'$, and $\mathbb{B}_{i-1}$ and $\mathbb{B}_i$ are adjacent for all $i \in [\ell]$. The *length* of the reconfiguration sequence is defined as ℓ.

Let $M = (E, \mathcal{B})$ be a matroid. The *dual* of M is a pair $M^* = (E, \{E \setminus B \mid B \in \mathcal{B}\})$, which also forms a matroid [16]. A *coloop* of a matroid M is an element $e \in E$ that belongs to all the bases of M, that is, $e \in B$ for all $B \in \mathcal{B}$. Let $M = (E, \mathcal{B})$ and $M' = (E', \mathcal{B}')$ be matroids and let $\mathcal{B}^*$ be the family of maximal sets in $\{B \cup B' \mid B \in \mathcal{B}, B' \in \mathcal{B}'\}$. Then, the pair $(E \cup E', \mathcal{B}^*)$ is the *matroid union* of M and M', which is denoted $M \vee M'$. It is well known that $M \vee M'$ is also a matroid [16]. We can generalize this definition for more than two matroids: For k matroids $M_1, \ldots, M_k$, the matroid union of $M_1, \ldots, M_k$ is denoted by $\bigvee_{i=1}^k M_i$. If the ground sets E and E' of M and M' are disjoint, then $M \vee M'$ is called the *direct sum* of M and M', and we write $M \oplus M'$ instead of $M \vee M'$.

In our proofs, we use certain matroids. Let E be a finite set. For an integer r with $0 \leq r \leq |E|$, the *rank-r uniform matroid* on E is the pair $(E, \{B \subseteq E \mid |B| = r\})$, that is, the set of bases consists of all size-r subsets of E. Let $\{E_1, \ldots, E_k\}$ be a partition of E (i.e., $E = \bigcup_{i=1}^k E_i$ and $E_i \cap E_j = \emptyset$ for distinct $i, j \in [k]$). For each $i \in [k]$, we set r_i as an integer with $0 \leq r_i \leq |E_i|$. If $\mathcal{B} \subseteq 2^E$ consists of the sets B satisfying $|B \cap E_i| = r_i$ for each $i \in [k]$, then the pair $(E, \mathcal{B})$ forms a matroid, called the *partition matroid*. We can construct such a partition matroid by taking the direct sum of the rank-r_i uniform matroids on E_i for i.

Let $D = (V, A)$ be a directed graph. For an arc $a \in A$, we write $\mathrm{head}(a)$ to denote the head of e and $\mathrm{tail}(a)$ to denote the tail of e. A *matching* of D is a set $N \subseteq A$ of arcs such that no pair of arcs in N share a vertex. A *walk* in D is a sequence $(v_0, a_1, v_1, a_2, \ldots, a_\ell, v_\ell)$ such that $\mathrm{tail}(a_i) = v_{i-1}$ and $\mathrm{head}(a_i) = v_i$ for all $i \in [\ell]$. When no confusion is possible, we may identify the directed graph with its arc set.

3 Polynomial-time algorithm

This section is devoted to a polynomial-time algorithm for Basis Sequence Reconfiguration, implying Theorem 1. Let $M_1 = (E_1, \mathcal{B}_1), M_2 = (E_2, \mathcal{B}_2), \ldots, M_k = (E_k, \mathcal{B}_k)$ be k matroids that are given as basis oracles. We denote by $\mathbb{M} = (M_1, M_2, \ldots, M_k)$ the tuple of matroids $M_1, \ldots, M_k$.

Let $\mathbb{B} = (B_1, \ldots, B_k)$ and $\mathbb{B}' = (B'_1, \ldots, B'_k)$ be two feasible basis sequences of $\mathbb{M}$. Take any coloop x of the matroid union $\bigvee_{i=1}^k M_i$. Since all bases in $\mathbb{B}$ are mutually disjoint, $\bigcup_{i=1}^k B_i$ is a basis of $\bigvee_{i=1}^k M_i$. This implies that $x \in B_i$ for some i. Suppose that there is a feasible basis sequence $(B_1, \ldots, B_{i-1}, B_i - x + y, B_{i+1}, \ldots, B_k)$ of $\mathbb{M}$ obtained from $\mathbb{B}$

[2] Our algorithm also runs in polynomial time even when the input matroids are given as *independence* or *rank* oracles.

by exchanging $x \in B_i$ with $y \in E_i \setminus B_i$ in M_i. As it is feasible, $\bigcup_{i=1}^{k} B_i - x + y$ is also a basis of $\bigvee_{i=1}^{k} M_i$, contradicting the fact that x is a coloop. This implies that every coloop in $\bigvee_{i=1}^{k} M_i$ belongs to a basis in a feasible basis sequence that is reconfigurable from $\mathbb{B}$. More formally, let K denote the set of coloops in $\bigvee_{i=1}^{k} M_i$. If $\mathbb{B}$ is reconfigurable to $\mathbb{B}'$, we have $(K \cap B_1, \dots, K \cap B_k) = (K \cap B'_1, \dots, K \cap B'_k)$. The following theorem says that this necessary condition is also sufficient.

▶ **Theorem 3.** *Let K be the set of coloops of $\bigvee_{i=1}^{k} M_i$. For feasible basis sequences $\mathbb{B} = (B_1, \dots, B_k)$ and $\mathbb{B}' = (B'_1, \dots, B'_k)$ of $\mathbb{M}$, one is reconfigurable to the other if and only if $(K \cap B_1, \dots, K \cap B_k) = (K \cap B'_1, \dots, K \cap B'_k)$.*

The proof of Theorem 3 is given in Section 3.2 below. Before the proof, we introduce the concept of *exchangeability graphs* and present its properties in Section 3.1.

3.1 Exchangeability graph

For a matroid $M = (E, \mathcal{B})$ and a basis $B \in \mathcal{B}$, the *exchangeability graph* of M with respect to B, denoted as $D(M, B)$, is a directed graph whose vertex set is the ground set E of M and whose arc set A is

$$A := \{(x, y) \mid x \in B \text{ and } y \in E \setminus B \text{ such that } B - x + y \in \mathcal{B}\}.$$

Note that $D(M, B)$ is bipartite; all arcs go from B to $E \setminus B$.

Let $N = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \subseteq A$ be a matching of $D(M, B)$ and let $B \triangle N := B \setminus \{x_1, x_2, \dots, x_n\} \cup \{y_1, y_2, \dots, y_n\}$. We say that N is *unique* if there is no perfect matching N' other than N in the subgraph of $D(M, B)$ induced by $\{x_1, \dots, x_n, y_1, \dots, y_n\}$. The following is a well-known lemma in matroid theory, called the *unique-matching lemma.*

▶ **Lemma 4** (e.g., [14, Lemma 2.3.18]). *If $N = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ is a unique matching in the subgraph of $D(M, B)$ induced by $\{x_1, \dots, x_n, y_1, \dots, y_n\}$, then $B \triangle N \in \mathcal{B}$.*

The *exchangeability graph* of $\mathbb{M}$ with respect to $\mathbb{B}$, denoted as $D(\mathbb{M}, \mathbb{B})$, is the union of the exchangeability graphs $D(M_i, B_i) = (E_i, A_i)$ of M_i with respect to B_i for all $i \in [k]$. In the following, the vertex set of $D(\mathbb{M}, \mathbb{B})$ is denoted by E, that is, $E = \bigcup_{i=1}^{k} E_i$. We note that, for distinct $i, j \in [k]$, the two arc sets A_i and A_j are disjoint, since $B_i \cap B_j = \emptyset$. A walk W in $D(\mathbb{M}, \mathbb{B})$ is called a *tadpole-walk* if W is of the form

$$(x_0, a_1, x_1, \dots, x_{m-1}, a_m, x_m = x_0, a_{m+1}, x_{m+1}, a_{m+2}, \dots, a_n, x_n) \tag{1}$$

for some $0 \le m < n$ such that the former part $(x_0, a_1, x_1, \dots, x_{m-1}, a_m, x_m = x_0)$ forms a directed cycle and the latter part $(x_m = x_0, a_{m+1}, x_{m+1}, \dots, x_n)$ forms a directed path with $x_n \in E \setminus \bigcup_{i=1}^{k} B_i$, where $x_0, x_1, \dots, x_n$ are distinct except for $x_0 = x_m$ if $m > 0$. See Figure 2 for an illustration. The former part can be empty; in this case, W is just a directed path ending at some vertex in $E \setminus \bigcup_{i=1}^{k} B_i$. We introduce a total order $\prec$ on the vertex set of W as: The smallest vertex is $x_0 (= x_m)$ and $x_i \prec x_j$ if and only if $i < j$ for other vertices x_i, x_j. We say that W is *shortcut-free* if, for all $i \in [k]$ and two arcs $a, a' \in W \cap A_i$ with $\mathrm{tail}(a) \prec \mathrm{tail}(a')$, we have $(\mathrm{tail}(a), \mathrm{head}(a')) \notin A_i$. A subgraph W' of $D(\mathbb{M}, \mathbb{B})$ is said to be *valid* if it is the disjoint union of a (possibly empty) directed path ending at some vertex in $E \setminus \bigcup_{i=1}^{k} B_i$ and a (possibly empty) directed cycle. For a valid subgraph W' of $D(\mathbb{M}, \mathbb{B})$, we define $\mathbb{B} \triangle W' := (B_1 \triangle (W' \cap A_1), B_2 \triangle (W' \cap A_2), \dots, B_k \triangle (W' \cap A_k))$. Observe first that $W' \cap A_i$ forms a matching in $D(M_i, B_i)$ for each i. To see this, suppose that there are two arcs a and a' in $W' \cap A_i$ that share a vertex x. Since each component of W' is either

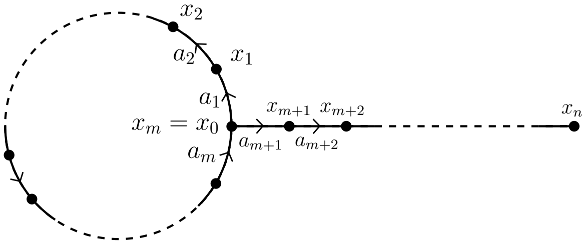


Figure 2 A tadpole-walk starting from x_0 and ending at x_n.

a directed path or a directed cycle, we can assume that $\mathrm{head}(a) = \mathrm{tail}(a') = x$. However, $x \notin B_i$ as $\mathrm{head}(a) = x$ and $x \in B_i$ as $\mathrm{tail}(a') = x$, a contradiction. Observe next that $|\bigcup_{i=1}^k B_i| = |\bigcup_{i=1}^k (B_i \triangle (W' \cap A_i))|$. This follows from the fact that the path component has a sink vertex in $E \setminus \bigcup_{i=1}^k B_i$ (if it is nonempty).

The following two lemmas play important roles in the proof of Theorem 3.

▶ **Lemma 5.** *Suppose that W is a shortcut-free tadpole-walk in $D(\mathbb{M}, \mathbb{B})$ and W' is a valid subgraph of W. Then $\mathbb{B} \triangle W'$ is a feasible basis sequence of $\mathbb{M}$.*

Proof. We first observe that $B_i \triangle (W' \cap A_i)$ and $B_j \triangle (W' \cap A_j)$ are disjoint for distinct $i, j \in [k]$. This follows from the following facts: $|B_i| = |B_i \triangle (W' \cap A_i)|$ for each i and $|\bigcup_{i=1}^k B_i| = |\bigcup_{i=1}^k (B_i \triangle (W' \cap A_i))|$. Thus, it suffices to show that $B_i \triangle (W' \cap A_i) \in \mathcal{B}_i$ for each i.

Let $b_1, b_2, \ldots, b_\ell$ be the arcs in the matching $W' \cap A_i$; we may assume that $i < j$ if and only if $\mathrm{tail}(b_i) \prec \mathrm{tail}(b_j)$. Since W is shortcut-free, we have $(\mathrm{tail}(b_i), \mathrm{head}(b_j)) \notin A_i$ for any distinct $i, j \in [\ell]$ with $i < j$. Observe that $W' \cap A_i$ forms a unique matching in $D(M_i, B_i)$. This can be seen by considering the other case that $W' \cap A_i$ is not unique in $D(M_i, B_i)$, yielding that $D(M_i, B_i)$ has an arc $(\mathrm{tail}(b_i), \mathrm{head}(b_j))$ for some $i, j \in [\ell]$ with $i < j$. Thus $B_i \triangle (W' \cap A_i) \in \mathcal{B}_i$ by Lemma 4. ◀

▶ **Lemma 6.** *Let $\mathbb{B} = (B_1, \ldots, B_k)$ be a feasible basis sequence of $\mathbb{M}$ and $B := \bigcup_{i=1}^k B_i$. For $y \in E \setminus B$, we denote by T_y the set of vertices that are reachable to y in $D(\mathbb{M}, \mathbb{B})$, that is, the set of vertices x in E such that there is a directed path from x to y in $D(\mathbb{M}, \mathbb{B})$. Then the set of coloops of $M := \bigvee_{i=1}^k M_i$ is equal to $B \setminus \bigcup_{y \in E \setminus B} T_y$.*

Proof. Clearly B contains all coloops of M as B is a basis of M. By considering the basis exchange axiom for the dual matroid M^* of M, an element $x \in B$ is not a coloop of M if and only if there is $y \in E \setminus B$ such that $B - x + y$ is a basis of M. Here, it easily follows from [19, Theorem 42.4] that, for $x \in B$ and $y \in E \setminus B$, the set $B - x + y$ is a basis of M if and only if there is a directed path from x to y in $D(\mathbb{M}, \mathbb{B})$. Hence the existence of such $y \in E \setminus B$ can be rephrased as the existence of a directed path from x to some vertex $y \in E \setminus B$ in $D(\mathbb{M}, \mathbb{B})$. This implies that the set of coloops of M is equal to $B \setminus \bigcup_{y \in E \setminus B} T_y$, where T_y denotes the set of vertices that are reachable to y in $D(\mathbb{M}, \mathbb{B})$. ◀

Using Lemma 6, we can decide in polynomial time whether the condition $(K \cap B_1, \ldots, K \cap B_k) = (K \cap B'_1, \ldots, K \cap B'_k)$ in Theorem 3 holds as follows. Let $E = \bigcup_{i=1}^k E_i$. We can construct the exchangeability graph $D(\mathbb{M}, \mathbb{B})$ with $\sum_{i=1}^k |E_i|^2 \leq k|E|^2$ oracle calls. By Lemma 6, we can compute the set K of coloops of M in time $O(|E|^2)$ using a standard graph search algorithm.

3.2 Proof of Theorem 3

In this subsection, we provide the proof of Theorem 3, and then we also see that Theorem 1 follows from our proof of Theorem 3.

We define the distance $d(\mathbb{B}, \mathbb{B}')$ between $\mathbb{B}$ and $\mathbb{B}'$ by $d(\mathbb{B}, \mathbb{B}') := \sum_{i=1}^{k} |B_i \triangle B'_i|$. As we have already seen the only-if part of Theorem 3 in the previous subsection, in the following, we show the if part by induction on $d(\mathbb{B}, \mathbb{B}')$.

It is easy to see that $d(\mathbb{B}, \mathbb{B}') = 0$ if and only if $\mathbb{B} = \mathbb{B}'$. Suppose that $d(\mathbb{B}, \mathbb{B}') > 0$. If there is a feasible basis sequence $\mathbb{B}'' = (B''_1, B''_2, \ldots, B''_k)$ of $\mathbb{M}$ such that $\mathbb{B}$ is reconfigurable to $\mathbb{B}''$ and $d(\mathbb{B}'', \mathbb{B}') < d(\mathbb{B}, \mathbb{B}')$, we have $(B'_1 \cap K, \ldots, B'_k \cap K) = (B_1 \cap K, \ldots, B_k \cap K) = (B''_1 \cap K, \ldots, B''_k \cap K)$. Hence $\mathbb{B}''$ is reconfigurable to $\mathbb{B}'$ by induction, which implies that $\mathbb{B}$ is reconfigurable to $\mathbb{B}'$. Thus, our goal is to find such a feasible basis sequence $\mathbb{B}''$. To this end, we first compute a shortcut-free tadpole-walk W in $D(\mathbb{M}, \mathbb{B})$ and then transform $\mathbb{B}$ to $\mathbb{B}''$ one-by-one along this W. A crucial observation in this transformation is that each intermediate basis sequence is of the form $\mathbb{B} \triangle W'$ for some valid subgraph W' of W, meaning that it is a feasible basis sequence of $\mathbb{M}$ by Lemma 5.

Take any $x_0 \in \bigcup_{i=1}^{k} B_i \setminus B'_i$, say, $x_0 \in B_{i_0} \setminus B'_{i_0}$. Then there is $x_1 \in B'_{i_0} \setminus B_{i_0}$ such that $B_{i_0} - x_0 + x_1 \in \mathcal{B}_{i_0}$. Hence we have $a_1 = (x_0, x_1) \in A_{i_0}$. If $x_1 \in E \setminus \bigcup_{i=1}^{k} B$, we obtain a tadpole-walk (x_0, a_1, x_1); we are done. Otherwise, this vertex x_1 belongs to B_{i_1} for some i_1 $(\neq i_0)$. In particular, by $x_1 \in B'_{i_0}$, we have $x_1 \in B_{i_1} \setminus B'_{i_1}$. Hence there is $x_2 \in B'_{i_1} \setminus B_{i_1}$ such that $B_{i_1} - x_1 + x_2 \in \mathcal{B}_{i_1}$, implying $a_2 = (x_1, x_2) \in A_{i_1}$. By repeating this argument, we can find either of the following subgraphs of $D(\mathbb{M}, \mathbb{B})$:

Type I: a directed path $(x_0, a_1, x_1, \ldots, a_n, x_n)$ satisfying that $x_n \in E \setminus \bigcup_{i=1}^{k} B_i$ and that $x_\ell \in B_{i_\ell} \setminus B'_{i_\ell}$, $x_{\ell+1} \in B'_{i_\ell} \setminus B_{i_\ell}$, and $a_{\ell+1} \in A_{i_\ell}$ for all $\ell \in [0, n-1]$.

Type II: a directed cycle $(x_p, a_{p+1}, x_{p+1}, \ldots, x_{q-1}, a_q, x_q = x_p)$ satisfying that $x_\ell \in B_{i_\ell} \setminus B'_{i_\ell}$, $x_{\ell+1} \in B'_{i_\ell} \setminus B_{i_\ell}$, and $a_{\ell+1} \in A_{i_\ell}$ for all $\ell \in [p, q-1]$.

In the former case (Type I), the resulting directed path is a tadpole-walk. Consider the latter case (Type II). By the assumption that $(B_1 \cap K, \ldots, B_k \cap K) = (B'_1 \cap K, \ldots, B'_k \cap K)$, none of the vertices $x_p, \ldots, x_q$ belongs to the set K of coloops. This implies that, by Lemma 6, $D(\mathbb{M}, \mathbb{B})$ has a directed path from each vertex in the cycle to a vertex in $E \setminus \bigcup_{i=1}^{k} B_i$. We can choose such a directed path $(x_r, b_1, y_1, \ldots, b_m, y_m)$ from a vertex x_r in the cycle to a vertex y_m in $E \setminus \bigcup_{i=1}^{k} B_i$ so that the path is arc-disjoint from the cycle, by taking a shortest one among all such paths. Then, the walk $(x_r, a_r, x_{r+1}, \ldots, x_r, b_1, y_1, \ldots, b_m, y_m)$ forms a tadpole-walk. We denote by W the tadpole-walk obtained in these ways (Type I and II). In the following, by rearranging the indices, we may always assume that W is of the form (1), where the former part $C = (x_0, a_1, x_1, \ldots, x_{m-1}, a_m, x_m = x_0)$ is a directed cycle and the later part $P = (x_m = x_0, a_{m+1}, x_{m+1}, \ldots, x_n)$ is a directed path in $D(\mathbb{M}, \mathbb{B})$. Note that the directed cycle C can be empty, which corresponds to Type I.

We next update the above W so that W becomes shortcut-free. Suppose that W is not shortcut-free. Then there are arcs $a, a' \in W \cap A_i$ such that $\mathrm{tail}(a) \prec \mathrm{tail}(a')$ satisfying $a'' := (\mathrm{tail}(a), \mathrm{head}(a')) \in A_i$. Let $a = (x_p, x_{p+1})$ and let $a' = (x_q, x_{q+1})$. Since $W \cap A_i$ is a matching in $D(\mathbb{M}, \mathbb{B})$, we have $p + 1 \neq q$. We then execute one of the following update procedure:

- If a and a' belong to the directed path P, then update W as

$$W \leftarrow (x_0, \ldots, x_{m-1}, a_m, x_m = x_0, a_{m+1}, \ldots, a_p, x_p, a'', x_{q+1}, \ldots, x_n).$$

- If a and a' belong to the directed cycle C, then update W as

$$W \leftarrow (x_0, a_1, \ldots, a_p, x_p, a'', x_{q+1}, \ldots, x_m = x_0, a_{m+1}, x_{m+1}, \ldots, x_n).$$

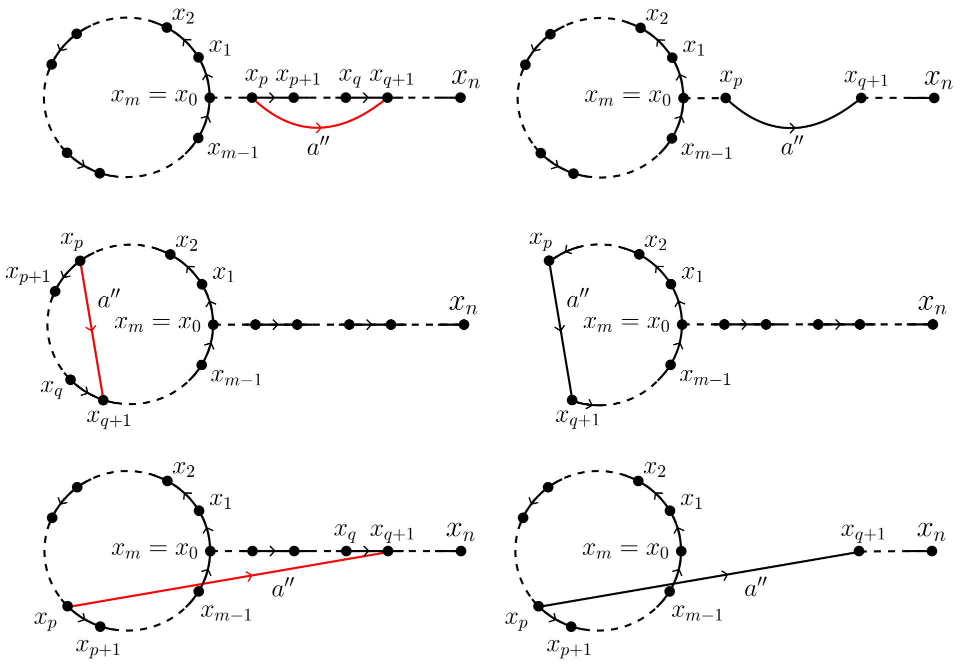


Figure 3 The figure depicts tadpole-walks (with shortcuts) and their updated tadpole-walks.

- If a belongs to C and a' belongs to P, then update W as

$$W \leftarrow (x_p, a_{p+1}, \ldots, a_p, x_p, a'', x_{q+1}, \ldots, x_n).$$

See Figure 3 for illustrations.

Suppose that W is a tadpole-walk of Type I. In this case, the second and third cases never occur. By the choice of $a = (x_p, x_{p+1})$, $a' = (x_q, x_{q+1})$, and $a'' = (x_p, x_{q+1})$, we have $x_p \in B_i \setminus B'_i$, $x_{q+1} \in B'_i \setminus B_i$, and $(x_p, x_{q+1}) \in A_i$. Moreover, the updated W is a directed path ending at $x_n \in E \setminus \bigcup_{i=1}^{k} B_i$, which implies that W is still a tadpole-walk of Type I. Suppose next that W is of Type II. In the first and third cases, the cycle part does not change; the updated W is still of a tadpole-walk Type II. In the second case, the cycle part is shortened by a'' but the updated W is still a tadpole-walk as well. By the choice of a, a', and a'', we have $x_p \in B_i \setminus B'_i$, $x_{q+1} \in B'_i \setminus B_i$, and $(x_p, x_{q+1}) \in A_i$. Hence the resulting W is still a tadpole-walk of Type II. Since this update procedure strictly reduces the size of W, we can eventually obtain a shortcut-free tadpole-walk in polynomial time.

Finally, we construct a reconfiguration sequence based on a shortcut-free tadpole-walk W of each type. Suppose that W is of Type I, i.e., $W = (x_0, a_1, x_1, \ldots, x_n)$ is a directed path with $n \geq 1$. For $p \in [n-1]$, let W_p denote the subgraph of W induced by $\{a_{n-p+1}, \ldots, a_n\}$. Then W_p forms a directed path $(x_{n-p}, a_{n-p+1}, x_{n-p+1}, \ldots, x_n)$, which implies that W_p is valid. By Lemma 5, $\mathbb{B} \triangle W_p$ is a feasible basis sequence for each p. Furthermore, we have $\mathbb{B} \triangle W_p = (\mathbb{B} \triangle W_{p-1}) \triangle (x_{n-p+1}, x_{n-p})$ (in which (x_{n-p+1}, x_{n-p}), the reverse of a_{n-p+1}, can be viewed as an arc in $D(\mathbb{M}, \mathbb{B} \triangle W_{p-1})$). This implies that $\mathbb{B} \triangle W_{p-1}$ and $\mathbb{B} \triangle W_p$ are adjacent for all $p \in [n-1]$, where $W_0 := \emptyset$. Hence

$$\langle \mathbb{B} = \mathbb{B} \triangle W_0, \mathbb{B} \triangle W_1, \mathbb{B} \triangle W_2, \ldots, \mathbb{B} \triangle W_{n-1} = \mathbb{B} \triangle W \rangle$$

is a reconfiguration sequence from $\mathbb{B}$ to $\mathbb{B} \triangle W$. In addition, since $x_\ell \in B_{i_\ell} \setminus B'_{i_\ell}$ and $x_{\ell+1} \in B'_{i_\ell} \setminus B_{i_\ell}$ for each $\ell \in [0, n-1]$, we have $d(\mathbb{B} \triangle W, \mathbb{B}') = d(\mathbb{B}, \mathbb{B}') - 2n < d(\mathbb{B}, \mathbb{B}')$.

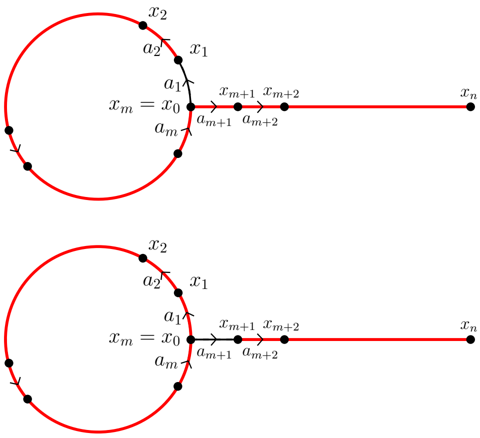


Figure 4 The bold red walk in the upper digraph represents W_p for $p = n-1$, and that in the lower digraph for $p = n$.

Suppose next that W is of Type II, i.e., W is of the form (1) with $0 < m < n$, where the (nonempty) former part $C = (x_0, a_1, x_1, \ldots, x_{m-1}, a_m, x_m = x_0)$ is a directed cycle and the later part $P = (x_m = x_0, a_{m+1}, x_{m+1}, \ldots, x_n)$ is a directed path. For $p \in [n-1]$, let W_p denote the subgraph of W induced by $\{a_{n-p+1}, \ldots, a_n\}$, which forms a directed path $(x_{n-p}, a_{n-p+1}, x_{n-p+1}, \ldots, x_n)$ as in the case of Type I and is valid. For $p \in [n, 2n-m-1]$, let W_p denote the subgraph of W induced by $\{a_1, a_2, \ldots, a_m\} \cup \{a_{(m-n+2)+p}, a_{(m-n+2)+(p+1)}, \ldots, a_n\}$, where W_{2n-m-1} is defined as C. In this case, W_p forms the disjoint union of the directed cycle C and the subpath of P starting from $x_{m-n+1+p}$ ending at $x_n \in E \setminus \bigcup_{i=1}^{k} B_i$; the subpath is empty if $p = 2n-m-1$. Thus W_p is also valid. By Lemma 5, $\mathbb{B} \triangle W_p$ is feasible for each $p \in [2n-m-1]$. Furthermore, we have

$$\mathbb{B} \triangle W_p = \begin{cases} (\mathbb{B} \triangle W_{p-1}) \triangle (x_{n-p+1}, x_{n-p}) & \text{if } p \in [n-1], \\ (\mathbb{B} \triangle W_{p-1}) \triangle (x_{m+1}, x_1) & \text{if } p = n, \\ (\mathbb{B} \triangle W_{p-1}) \triangle a_{(m-n+1)+p} & \text{if } p \in [n+1, 2n-m-1]. \end{cases}$$

See Figure 4 for the case of $p = n$. This implies that $\mathbb{B} \triangle W_{p-1}$ and $\mathbb{B} \triangle W_p$ are adjacent for all $p \in [2n-m-1]$, where $W_0 := \emptyset$. Hence

$$\langle \mathbb{B} = \mathbb{B} \triangle W_0, \mathbb{B} \triangle W_1, \mathbb{B} \triangle W_2, \ldots, \mathbb{B} \triangle W_{2n-m-1} = \mathbb{B} \triangle C \rangle$$

is a reconfiguration sequence from $\mathbb{B}$ to $\mathbb{B} \triangle C$. In addition, since $x_\ell \in B_{i_\ell} \setminus B'_{i_\ell}$ and $x_{\ell+1} \in B'_{i_\ell} \setminus B_{i_\ell}$ for each $\ell \in [m]$, we have $d(\mathbb{B} \triangle C, \mathbb{B}') = d(\mathbb{B}, \mathbb{B}') - 2m < d(\mathbb{B}, \mathbb{B}')$. This completes the proof of Theorem 3.

The above proof immediately turns into an algorithm for finding a feasible basis sequence $\mathbb{B}''$ with $d(\mathbb{B}'', \mathbb{B}') < d(\mathbb{B}, \mathbb{B}')$ in polynomial time. As shown in the previous subsection, we can construct the exchangeability graph $D(\mathbb{M}, \mathbb{B})$ using $k|E|^2$ oracle calls. We can compute a shortcut-free tadpole-walk in $D(\mathbb{M}, \mathbb{B})$ in $O(|E|^2)$ time. Thus, we can compute a feasible basis sequence $\mathbb{B}''$ of $\mathbb{M}$ with $d(\mathbb{B}'', \mathbb{B}') < d(\mathbb{B}, \mathbb{B}')$ such that $\mathbb{B}$ is reconfigurable to $\mathbb{B}''$ in $O(|E|^2)$ time and $|E|^2$ oracle calls. Since $d(\mathbb{B}, \mathbb{B}')$ is at most $2|E|$, we can obtain an entire reconfiguration sequence from $\mathbb{B}$ to $\mathbb{B}'$ in $O(|E|^3)$ time and $|E|^3$ oracle calls in the case where $\mathbb{B}$ is reconfigurable to $\mathbb{B}'$. Note that the length of the above reconfiguration sequence is $O(|E|^2)$. Therefore, Theorem 1 follows.

4 Inapproximability of finding a shortest reconfiguration sequence

In this section, we prove Theorem 2, that is, Shortest Basis Sequence Reconfiguration is hard to approximate in polynomial time under $\mathsf{P} \neq \mathsf{NP}$. To show this inapproximability result, we perform a reduction from Set Cover, which is notoriously hard to approximate.

Let $\mathcal{S} \subseteq 2^U$ be a family of subsets of a finite set U where $n = |U|$ and $m = |\mathcal{S}|$. We say that a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ *covers* U (or $\mathcal{S}'$ is a *set cover* of U) if $U = \bigcup_{S \in \mathcal{S}'} S$. Set Cover is the problem that, given a set U and a family $\mathcal{S} \subseteq 2^U$ of subsets of U, asks to find a minimum cardinality subfamily $\mathcal{S}' \subseteq \mathcal{S}$ that covers U. Set Cover is known to be hard to approximate: Raz and Safra [17] showed that there is a constant $c^* > 0$ such that it is NP-hard to find a $c^* \log(n+m)$-approximate solution of Set Cover. Throughout this section, we assume that the whole family $\mathcal{S}$ covers U.

From an instance $(U, \mathcal{S})$ of Set Cover, we construct two partition matroids $M_1 = (E_1, \mathcal{B}_1)$ and $M_2 = (E_2, \mathcal{B}_2)$ such that there is a set cover of U of size at most k if and only if there is a reconfiguration sequence between feasible basis sequences $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ of $\mathbb{M} = (M_1, M_2)$ with length at most ℓ for some ℓ.

4.1 Construction

To construct the partition matroids M_1 and M_2, we use several uniform matroids and combine them into M_1 and M_2. In the following, we assume that the sets in $\mathcal{S}$ are ordered in an arbitrary total order $\preceq$. For each element $u \in U$, we define $f(u) + 3$ elements $e_u^1, e_u^2, e_u^3, c_u^1, \ldots, c_u^{f(u)}$ and three sets:

$$E_u^1 \coloneqq \{e_u^1, e_u^2\}, \qquad E_u^2 \coloneqq \{e_u^1, e_u^2, e_u^3\}, \qquad E_u^3 \coloneqq \{e_u^3\} \cup \{c_u^1, c_u^2, \ldots, c_u^{f(u)}\},$$

where $f(u) \coloneqq |\{S \in \mathcal{S} \mid u \in S\}|$ is the number of occurrences of u in $\mathcal{S}$. We denote by M_u^i the rank-1 uniform matroid over E_u^i for $1 \leq i \leq 3$, that is, each basis of M_u^i contains exactly one element in E_u^i. For each set $S \in \mathcal{S}$, we denote by M_S^0 the uniform matroid of rank $|S|$ with ground set $E_S^0 \coloneqq \{c_u^{f(u,S)} \mid u \in S\} \cup \{s_S^1\}$, where $f(u, S) = |\{S' \mid S' \preceq S, u \in S'\}|$. Note that $E_S^0 \cap E_{S'}^0 = \emptyset$ for distinct $S, S' \in \mathcal{S}$. We let $L \coloneqq 2n^2$. For $1 \leq i \leq L$, we denote by M_S^i the rank-1 matroid with ground set $E_S^i \coloneqq \{s_S^i, s_S^{i+1}\}$. Then, we define two partition matroids M_1 and M_2 as:

$$M_1 \coloneqq \bigoplus_{u \in U} M_u^1 \oplus \bigoplus_{u \in U} M_u^3 \oplus \bigoplus_{S \in \mathcal{S}} \bigoplus_{i=1}^{n^2} M_S^{2i-1}, \qquad M_2 \coloneqq \bigoplus_{u \in U} M_u^2 \oplus \bigoplus_{S \in \mathcal{S}} M_S^0 \oplus \bigoplus_{S \in \mathcal{S}} \bigoplus_{i=1}^{n^2} M_S^{2i}.$$

The matroids M_1 and M_2 are illustrated in Figure 5. We denote by E_1 and E_2 the ground sets and by $\mathcal{B}_1$ and $\mathcal{B}_2$ the collections of bases of M_1 and M_2, respectively. Each uniform matroid constituting these partition matroids is called a *block*. Since M_1 and M_2 are partition matroids, the following observation follows.

▶ **Observation 7.** *Let (B_1, B_2) be a feasible basis sequence of (M_1, M_2) and let $x \in B_1$ be arbitrary. Then, for any element $y \in E_1 \setminus (B_1 \cup B_2)$ that belongs to the same block as x in M_1, $(B_1 - x + y, B_2)$ is a feasible basis sequence of (M_1, M_2). Similarly, let $x \in B_2$ be arbitrary. Then, for any element $y \in E_2 \setminus (B_1 \cup B_2)$ that belongs to the same block as x in M_2, $(B_1, B_2 - x + y)$ is a feasible basis sequence of (M_1, M_2).*

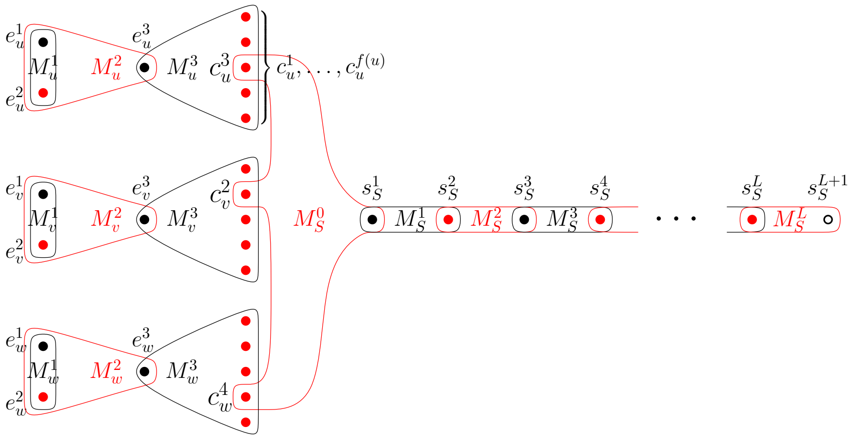


Figure 5 The figure depicts (hypergraph representations of) two partition matroids M_1 and M_2. A set $S \in \mathcal{S}$ contains three elements $u, v, w \in U$ with $f(u, S) = 3$, $f(v, S) = 2$, and $f(w, S) = 4$. Solid black circles represent elements in B_1^{s}, and solid red circles represent elements in B_2^{s}.

Let $\mathbb{B}^{\mathsf{s}} = (B_1^{\mathsf{s}}, B_2^{\mathsf{s}})$ be a feasible basis sequence such that

$$B_1^{\mathsf{s}} = \{e_u^1 \mid u \in U\} \cup \{e_u^3 \mid u \in U\} \cup \bigcup_{S \in \mathcal{S}} \{s_S^{2i-1} \mid i \in [n^2]\},$$

$$B_2^{\mathsf{s}} = \{e_u^2 \mid u \in U\} \cup \bigcup_{S \in \mathcal{S}} \{c_u^{f(u,S)} \mid u \in S\} \cup \bigcup_{S \in \mathcal{S}} \{s_S^{2i} \mid i \in [n^2]\}.$$

It is easy to verify that B_1^{s} and B_2^{s} are bases of M_1 and M_2, respectively. Similarly, let $\mathbb{B}^{\mathsf{t}} = (B_1^{\mathsf{t}}, B_2^{\mathsf{t}})$ be a feasible basis sequence such that

$$B_1^{\mathsf{t}} = \{e_u^2 \mid u \in U\} \cup \{e_u^3 \mid u \in U\} \cup \bigcup_{S \in \mathcal{S}} \{s_S^{2i-1} \mid i \in [n^2]\},$$

$$B_2^{\mathsf{t}} = \{e_u^1 \mid u \in U\} \cup \bigcup_{S \in \mathcal{S}} \{c_u^{f(u,S)} \mid u \in S\} \cup \bigcup_{S \in \mathcal{S}} \{s_S^{2i} \mid i \in [n^2]\}.$$

Let us note that $s_S^{L+1} \notin B_1^{\mathsf{s}} \cup B_2^{\mathsf{s}} \cup B_1^{\mathsf{t}} \cup B_2^{\mathsf{t}}$ for all $S \in \mathcal{S}$. Moreover, we have $B_1^{\mathsf{s}} \setminus B_1^{\mathsf{t}} = B_2^{\mathsf{t}} \setminus B_2^{\mathsf{s}} = \{e_u^1 \mid u \in U\}$ and $B_1^{\mathsf{t}} \setminus B_1^{\mathsf{s}} = B_2^{\mathsf{s}} \setminus B_2^{\mathsf{t}} = \{e_u^2 \mid u \in U\}$.

4.2 Correctness

Before proceeding to our proof, we first give the intuition behind our construction. Suppose that there are tokens on the elements in $B_1^{\mathsf{s}} \cup B_2^{\mathsf{s}}$. As observed in the previous subsection, we have $e_u^1 \in B_1^{\mathsf{s}} \setminus B_1^{\mathsf{t}}$ and $e_u^1 \in B_2^{\mathsf{t}} \setminus B_2^{\mathsf{s}}$ for $u \in U$. Moreover, $e_u^2 \in B_1^{\mathsf{t}} \setminus B_1^{\mathsf{s}}$ and $e_u^2 \in B_2^{\mathsf{s}} \setminus B_2^{\mathsf{t}}$ for $u \in U$. Thus, in order to transform $\mathbb{B}^{\mathsf{s}}$ to $\mathbb{B}^{\mathsf{t}}$, we need to “swap” the tokens on e_u^1 and e_u^2. However, as all the elements except for s_S^{L+1} for $S \in \mathcal{S}$ are occupied by tokens in $B_1^{\mathsf{s}} \cup B_2^{\mathsf{s}}$, this requires to move an “empty space” initially placed on s_S^{L+1} to e_u^3 for some $S \in \mathcal{S}$ with $u \in S$, and then swap the tokens on e_u^1 and e_u^2 using the empty space on e_u^3. By the construction of M_1 and M_2, this can be done by (1) shifting the tokens along the path between s_S^{L+1} and s_S^1 one by one, (2) moving the empty space from s_S^1 to $c_u^{f(u,S)}$, and then (3) moving the empty space from $c_u^{f(u,S)}$ to e_u^3, which requires at least L exchanges. As L is sufficiently large, we need to cover the elements in U with a small number of sets in $\mathcal{S}$ for a short reconfiguration sequence. The following lemma gives an upper bound on the length of a shortest reconfiguration sequence.

Algorithm 1 An algorithm for constructing a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ from a set cover $\mathcal{S}^* \subseteq \mathcal{S}$ of U.

```
   Input: A set cover $\mathcal{S}^* \subseteq \mathcal{S}$ of $U$.
   Output: A reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$.
1  $\tilde{U} \leftarrow \emptyset$, $\mathbb{B} \leftarrow \mathbb{B}^{\mathsf{s}}$
2  foreach $S \in \mathcal{S}^*$ do
3      for $i = L, L-1, \dots, 1$ do
4          $\mathbb{B} \leftarrow \mathbb{B} \triangle (s_S^i, s_S^{i+1})$
5  foreach $S \in \mathcal{S}^*$ do
6      if $S \setminus \tilde{U} \neq \emptyset$ then
7          foreach $u \in S \setminus \tilde{U}$ do
8              $\mathbb{B} \leftarrow \mathbb{B} \triangle (c_u^{f(u,S)}, s_S^1)$
9              $\mathbb{B} \leftarrow \mathbb{B} \triangle (e_u^3, c_u^{f(u,S)})$
10             $\mathbb{B} \leftarrow \mathbb{B} \triangle (e_u^2, e_u^3)$
11             $\mathbb{B} \leftarrow \mathbb{B} \triangle (e_u^1, e_u^2)$
12             $\mathbb{B} \leftarrow \mathbb{B} \triangle (e_u^3, e_u^1)$
13             $\mathbb{B} \leftarrow \mathbb{B} \triangle (c_u^{f(u,S)}, e_u^3)$
14             $\mathbb{B} \leftarrow \mathbb{B} \triangle (s_S^1, c_u^{f(u,S)})$
15         $\tilde{U} \leftarrow \tilde{U} \cup S$
16 foreach $S \in \mathcal{S}^*$ do
17     for $i = 1, 2, \dots, L$ do
18         $\mathbb{B} \leftarrow \mathbb{B} \triangle (s_S^{i+1}, s_S^i)$
```

▶ **Lemma 8.** *Let $\mathcal{S}^* \subseteq \mathcal{S}$ be a set cover of U of size at most k. Then, there is a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ with length at most $2kL + 7n$.*

Proof. Given a set cover $\mathcal{S}^* \subseteq \mathcal{S}$, we construct a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ by applying the algorithm described in Algorithm 1. Let $\mathbb{B} = (B_1, B_2)$ be a feasible basis sequence of (M_1, M_2). For $x, y \in E_1 \cup E_2$, we call (x, y) a *valid pair* if either

(1) $x \in B_1$ and $y \in E_1 \setminus (B_1 \cup B_2)$ belong to the same block in M_1; or

(2) $x \in B_2$ and $y \in E_2 \setminus (B_1 \cup B_2)$ belong to the same block in M_2.

For a valid pair (x, y), we define

$$\mathbb{B} \triangle (x, y) = \begin{cases} (B_1 - x + y, B_2) & \text{if } (x, y) \text{ satisfies (1)}, \\ (B_1, B_2 - x + y) & \text{if } (x, y) \text{ satisfies (2)}. \end{cases}$$

By Observation 7, $\mathbb{B} \triangle (x, y)$ is a feasible basis sequence of (M_1, M_2).

When we update a feasible basis sequence $\mathbb{B} = (B_1, B_2)$ with $\mathbb{B} \triangle (x, y)$ for some $x, y \in E_1 \cup E_2$ in the algorithm, the pair (x, y) is always assured to be valid. Thus, all the pairs $\mathbb{B} = (B_1, B_2)$ appearing in the execution of the algorithm are feasible basis sequences of (M_1, M_2). Since $\mathcal{S}^*$ is a set cover of U, we have $\tilde{U} = U$ when the algorithm terminates. Thus, for each $u \in U$, the steps from line 8 to line 14 are executed exactly once. This implies that the algorithm correctly computes a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ with length $2kL + 7n$. ◀

▶ **Lemma 9.** *Suppose that there is a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ of length ℓ. Then, there is a set cover $\mathcal{S}^* \subseteq \mathcal{S}$ of U with $|\mathcal{S}^*| \leq \lfloor \ell / 2L \rfloor$.*

Proof. Let $\sigma = \langle \mathbb{B}_0, \ldots, \mathbb{B}_\ell \rangle$ be a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ of length ℓ. For a feasible basis sequence $\mathbb{B} = (B_1, B_2)$, an element $e \in E_1 \cup E_2$ is said to be *free* in $\mathbb{B}$ if $e \notin B_1 \cup B_2$. We define $\mathcal{S}^* := \{S \in \mathcal{S} \mid s_S^1 \text{ is free in } \mathbb{B}_i \text{ for some } i\}$. Then the following holds.

▷ Claim 10. The subfamily $\mathcal{S}^*$ of $\mathcal{S}$ is a set cover of U.

Proof. Let $\mathbb{B}_i = (B_1^i, B_2^i)$ for $i \in [0, \ell]$. We first observe that, for $S \in \mathcal{S}$ and $i \in [0, \ell]$, if $s_S^1 \in B_1^i$, then $\{c_u^{f(u,S)} \mid u \in S\} \subseteq B_2^i$. This can be seen as follows. Since $s_S^1 \in B_1^i$, we have $s_S^1 \notin B_2^i$. As B_2^i must contain a basis B_S^0 of M_S^0, which is the uniform matroid of rank $|S|$ with the ground set $\{c_u^{f(u,S)} \mid u \in S\} \cup \{s_S^1\}$, the basis B_S^0 must be $\{c_u^{f(u,S)} \mid u \in S\}$. That is, we have $\{c_u^{f(u,S)} \mid u \in S\} \subseteq B_2^i$.

We then show the assertion of Claim 10. Suppose for contradiction that there is an element $u^* \in U$ that is not covered by $\mathcal{S}^*$. Then, for $S \in \mathcal{S}$ with $u^* \in S$, the element s_S^1 is not free in $\mathbb{B}_i$ for any $0 \le i \le \ell$, which implies that s_S^1 belongs to B_1^i. Thus, for each i, we have $B_2^i \supseteq \bigcup_{S \in \mathcal{S} : u^* \in S} \{c_u^{f(u,S)} \mid u \in S\} \supseteq \{c_{u^*}^1, \ldots, c_{u^*}^{f(u^*)}\}$, where the first inclusion follows from the above observation. By this inclusion with the fact that $M_{u^*}^3$ is the uniform matroid of rank 1 with the ground set $\{e_{u^*}^3\} \cup \{c_{u^*}^1, \ldots, c_{u^*}^{f(u^*)}\}$, the basis B_1^i must contain $e_{u^*}^3$ for each i. Hence, during the reconfiguration sequence $\sigma = \langle \mathbb{B}_0, \ldots, \mathbb{B}_\ell \rangle$, we cannot move any element in $E_{u^*}^1 = \{e_{u^*}^1, e_{u^*}^2\}$ (or more precisely $E_{u^*}^1 \cup E_{u^*}^2 \cup E_{u^*}^3$). This contradicts that σ is a reconfiguration sequence from $\mathbb{B}^{\mathsf{s}}$ to $\mathbb{B}^{\mathsf{t}}$; recall $e_{u^*}^1 \in B_1^{\mathsf{s}} \setminus B_1^{\mathsf{t}} = B_2^{\mathsf{t}} \setminus B_2^{\mathsf{s}}$ and $e_{u^*}^2 \in B_1^{\mathsf{t}} \setminus B_1^{\mathsf{s}} = B_2^{\mathsf{s}} \setminus B_2^{\mathsf{t}}$. ◁

In the reconfiguration sequence $\sigma = \langle \mathbb{B}_0, \ldots, \mathbb{B}_\ell \rangle$, for each $S \in \mathcal{S}^*$, the element s_S^{L+1} must be free in $\mathbb{B}_0$ and $\mathbb{B}_\ell$, and s_S^1 must be free at least once. Hence, the length ℓ of σ is at least $2L \cdot |\mathcal{S}^*|$, where L is equal to the number of required steps to move from a feasible basis sequence such that s_S^{L+1} (resp. s_S^1) is free to another feasible basis sequence such that s_S^1 (resp. s_S^{L+1}) is free. Since $\mathcal{S}^*$ is a set cover by Claim 10, we can conclude that there is a set cover of size at most $\lfloor \ell / 2L \rfloor$. ◀

Proof of Theorem 2. To prove the NP-hardness, we give a polynomial-time reduction from Set Cover. We claim that $I = (U, \mathcal{S})$ has a set cover of size at most k if and only if there is a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ of length at most $(2k+1) \cdot L$. We may assume $n \ge 4$.

Suppose that I has a set cover of size at most k. Then, by Lemma 8 and $7n \le 2n^2 = L$, we can construct a reconfiguration sequence from $\mathbb{B}^{\mathsf{s}}$ to $\mathbb{B}^{\mathsf{t}}$ of length at most $2kL + 7n \le 2kL + 2n^2 = (2k+1) \cdot L$, proving the forward implication.

Conversely, assume that there is a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$ of length at most $(2k+1) \cdot L$. Then, by Lemma 9, we obtain a set cover for I of the size at most $\lfloor (2k+1) \cdot L / 2L \rfloor = \lfloor k + 1/2 \rfloor = k$.

To prove the inapproximability, let $N = \sum_{i=1}^{k} |E_i|$ and suppose that there exists a $c' \log N$-approximation algorithm $\mathcal{A}'$ for Shortest Basis Sequence Reconfiguration for some constant $c' > 0$. Then we construct an algorithm $\mathcal{A}$ that, given an instance $I = (U, \mathcal{S})$ of Set Cover, outputs a set cover of I as follows.

1. Construct an instance $I' = (\mathbb{M}, \mathbb{B}^{\mathsf{s}}, \mathbb{B}^{\mathsf{t}})$ of Shortest Basis Sequence Reconfiguration from an instance $I = (U, \mathcal{S})$ of Set Cover using the construction in Section 4.1.
2. Compute a reconfiguration sequence σ' of I' by applying $\mathcal{A}'$.
3. Compute a set cover $\mathcal{S}^*$ for I from σ' by Lemma 9.

▷ Claim 11. For some constant $c > 0$, algorithm $\mathcal{A}$ produces a $c \log(n+m)$-approximation solution for Set Cover.

Proof. Since $\mathcal{S}$ covers U, by Lemma 8, there is a reconfiguration sequence between $\mathbb{B}^{\mathsf{s}}$ and $\mathbb{B}^{\mathsf{t}}$) of length at most $2L \cdot \mathrm{OPT}(I) + 7n$, where $\mathrm{OPT}(I)$ is the minimum cardinality of a set cover of U. Moreover, we have $N \leq (n+m)^d$ for some constant d. Thus, $\mathcal{A}'$ outputs a reconfiguration sequence σ' of length at most $\ell := c' \log N \cdot (2L \cdot \mathrm{OPT}(I) + 7n)$ in time $(n+m)^{O(1)}$. Finally, by Lemma 9, we can compute a set cover $\mathcal{S}^* \subseteq \mathcal{S}$ of U from σ' with size at most $\ell/2L = c' \log N \cdot (\mathrm{OPT}(I) + o(1)) \leq 2c' \log N \cdot \mathrm{OPT}(I)$. Since $N \leq (n+m)^d$, we have $|\mathcal{A}(I)| \leq c \log(n+m) \cdot \mathrm{OPT}(I)$ for any constant $c > 2c'd$. ◁

By choosing the constant c' as $c' < c^*/2d$, we derive a polynomial-time $c^* \log(n+m)$-approximation algorithm for Set Cover, completing the proof of Theorem 2. ◀

5 Conclusion

In this paper, we studied Basis Sequence Reconfiguration, which is a generalization of Spanning Tree Sequence Reconfiguration. For this problem, we first showed that Basis Sequence Reconfiguration can be solved in polynomial time, assuming that the input matroids are given as basis oracles. Second, we showed that the shortest variant of Basis Sequence Reconfiguration is hard to approximate within a factor of $c \log n$ for some constant $c > 0$ unless $\mathsf{P} = \mathsf{NP}$.

For future work, it is interesting to investigate the computational complexity of the special settings of Basis Sequence Reconfiguration. It would be interesting to design faster or simpler algorithms for Basis Sequence Reconfiguration with graphic matroids, that is, for Spanning Tree Sequence Reconfiguration. Our hardness result for the shortest variant uses two distinct partition matroids. Thus, it would be worth considering the case for two identical matroids. Finally, the computational complexity of Shortest Spanning Tree Sequence Reconfiguration is another promising direction.

References

1 Kristóf Bérczi, Bence Mátravölgyi, and Tamás Schwarcz. Reconfiguration of basis pairs in regular matroids. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1653–1664. ACM, 2024. doi:10.1145/3618260.3649660.

2 Jonah Blasiak. The toric ideal of a graphic matroid is generated by quadrics. *Comb.*, 28(3):283–297, 2008. doi:10.1007/S00493-008-2256-6.

3 Marthe Bonamy and Nicolas Bousquet. Recoloring graphs via tree decompositions. *Eur. J. Comb.*, 69:200–213, 2018. doi:10.1016/J.EJC.2017.10.010.

4 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: Pspace-completeness and superpolynomial distances. *Theor. Comput. Sci.*, 410(50):5215–5226, 2009. doi:10.1016/J.TCS.2009.08.023.

5 Nicolas Bousquet, Felix Hommelsheim, Yusuke Kobayashi, Moritz Mühlenthaler, and Akira Suzuki. Feedback vertex set reconfiguration in planar graphs. *Theor. Comput. Sci.*, 979:114188, 2023. doi:10.1016/J.TCS.2023.114188.

6 Martin Farber, B. Richter, and H. Shank. Edge-disjoint spanning trees: A connectedness theorem. *J. Graph Theory*, 9(3):319–324, 1985. doi:10.1002/JGT.3190090303.

7 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes. *IEICE Trans. Inf. Syst.*, 102-D(3):423–429, 2019. doi:10.1587/TRANSINF.2018FCP0005.

8 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

9 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.

10 Takehiro Ito, Yuni Iwamasa, Naonori Kakimura, Yusuke Kobayashi, Shun-ichi Maezawa, Yuta Nozaki, Yoshio Okamoto, and Kenta Ozeki. Rerouting planar curves and disjoint paths. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *LIPIcs*, pages 81:1–81:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.81.

11 Takehiro Ito, Yuni Iwamasa, Naoyuki Kamiyama, Yasuaki Kobayashi, Yusuke Kobayashi, Shun-ichi Maezawa, and Akira Suzuki. Reconfiguration of time-respecting arborescences. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS 2023)*, volume 14079 of *Lecture Notes in Computer Science*, pages 521–532. Springer, 2023. doi:10.1007/978-3-031-38906-1_34.

12 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, and Kunihiro Wasa. Reconfiguring (non-spanning) arborescences. *Theor. Comput. Sci.*, 943:131–141, 2023. doi:10.1016/J.TCS.2022.12.007.

13 Yusuke Kobayashi, Ryoga Mahara, and Tamás Schwarcz. Reconfiguration of the union of arborescences. In *Proceedings of the 34th International Symposium on Algorithms and Computation, (ISAAC 2023)*, volume 283 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.48.

14 Kazuo Murota. *Matrices and Matroids for Systems Analysis*. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-03994-2.

15 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

16 James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. Oxford University Press, Inc., USA, 2006.

17 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC 1997)*, pages 475–484. ACM, 1997. doi:10.1145/258533.258641.

18 Rin Saito, Hiroshi Eto, Takehiro Ito, and Ryuhei Uehara. Reconfiguration of vertex-disjoint shortest paths on graphs. In *Proceedings of the 17th International Conference and Workshops on Algorithms and Computation (WALCOM 2023)*, volume 13973 of *Lecture Notes in Computer Science*, pages 191–201. Springer, 2023. doi:10.1007/978-3-031-27051-2_17.

19 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Heidelberg, 2003.

20 Neil L. White. A unique exchange property for bases. *Linear Algebra and its Applications*, 31:81–91, 1980. doi:10.1016/0024-3795(80)90209-8.

Core Stability in Additively Separable Hedonic Games of Low Treewidth

Tesshu Hanaka
Kyushu University, Fukuoka, Japan

Noleen Köhler
University of Leeds, UK

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

Additively Separable Hedonic Games (ASHGs) are coalition-formation games where we are given a directed graph whose vertices represent n selfish agents and the weight of each arc uv denotes the preferences from u to v. We revisit the computational complexity of the well-known notion of core stability of symmetric ASHGs, where the goal is to construct a partition of the agents into coalitions such that no group of agents would prefer to diverge from the given partition and form a new coalition. For Core Stability Verification (CSV), we first show the following hardness results: CSV remains coNP-complete on graphs of vertex cover 2; CSV is coW[1]-hard parameterized by vertex integrity when edge weights are polynomially bounded; and CSV is coW[1]-hard parameterized by tree-depth even if all weights are from $\{-1, 1\}$. We complement these results with essentially matching algorithms and an FPT algorithm parameterized by the treewidth $\mathtt{tw}$ plus the maximum degree Δ (improving a previous algorithm's dependence from $2^{O(\mathtt{tw}\Delta^2)}$ to $2^{O(\mathtt{tw}\Delta)}$). We then move on to study Core Stability (CS), which one would naturally expect to be even harder than CSV. We confirm this intuition by showing that CS is Σ_2^p-complete even on graphs of bounded vertex cover number. On the positive side, we present a $2^{2^{O(\Delta\mathtt{tw})}} n^{O(1)}$-time algorithm parameterized by $\mathtt{tw} + \Delta$, which is essentially optimal assuming Exponential Time Hypothesis (ETH). Finally, we consider the notion of k-core stability: k denotes the maximum size of the allowed blocking (diverging) coalitions. We show that k-CSV is coW[1]-hard parameterized by k (even on unweighted graphs), while k-CS is NP-complete for all $k \geq 3$ (even on graphs of bounded degree with bounded edge weights).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Hedonic games, Treewidth, Core stability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.39

Related Version *Full Version*: `https://arxiv.org/abs/2402.10815`

Funding *Tesshu Hanaka*: Partially supported by JSPS KAKENHI Grant Numbers JP21H05852, JP21K17707, JP22H00513, JP23H04388.
Noleen Köhler: Partially supported by ANR project ANR-18-CE40-0025-01 (ASSK).
Michael Lampis: Supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

1 Introduction

Coalition-formation games model situations where a subset of selfish agents need to be partitioned into teams (coalitions) in such a way that takes into account their preferences. Because such games capture a vast array of interesting situations in the real world [40], they have been a subject of intense study in computational social choice and the social sciences at large. One particularly interesting and natural special case of such games is when the preferences of each agent only depend on the other agents that she is placed together with in

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 39; pp. 39:1–39:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the same coalition (and not on the placement of agents on other coalitions). Such games are referred to in the literature as *hedonic games* and have also attracted much interest from the computer science perspective [1, 2, 6, 7, 10, 12, 13, 20, 30, 35, 42], thanks in part to their numerous applications in, for example, social network analysis [36], scheduling group activities [18], and allocating tasks to wireless agents [41]. For more information we refer the reader to [14] and the relevant chapters of standard computational social choice texbooks [4].

Hedonic games are extremely general. Unfortunately, this generality renders them hard to study from the computer science perspective – indeed, even listing the preferences of all n agents takes space exponential in n as the naïve approach would require listing an ordering of all coalitions for each agent. This motivates the study of natural restrictions of hedonic games. In this paper we focus on one of the most natural such restrictions: *Additively Separable Hedonic Games* (ASHGs) introduced in [11], where the input is an edge-weighted directed graph, vertices represent the agents, and the weight of the arc uv denotes the preference of u for v, that is, the utility that agent u derives from being in the same coalition as v. The utility of an agent u in a coalition C can then be succinctly encoded as the sum of the weights of edges incident on u with their other endpoint in C. If the weights of uv and vu are the same for every pair of agents u, v, ASHGs are called *symmetric*. For symmetric ASHGs, the input graph becomes undirected.

In any situation where agents behave selfishly, it becomes critical to look for *stable* outcomes, that is, outcomes which the agents are likely to accept, based on their preferences. In the context of ASHGs, the question then becomes: given an edge-weighted graph G representing the agents' preferences, can we find a stable partition of the agents into coalitions (possibly also optimizing some other social welfare goal)? The computational complexity of such questions has been amply studied [3, 5, 21, 27, 36, 37, 44] and several natural notions of stability have been proposed. In this paper we revisit the computational complexity of one of the most widely-studied such notions, which is called *core stability* ([35, 39, 43, 45]). Intuitively, a partition of n agents is called *core stable*, if a group of agents does not want to leave their coalitions to form a coalition together. More formally, given a partition $\mathcal{P}$ of the agents, a *blocking coalition* is a set of agents X such that all $v \in X$ have strictly higher utility in X than in the initial partition $\mathcal{P}$. Hence, if a blocking coalition X exists, the initial partition is *unstable*, because the agents of X would prefer to form a new coalition. A partition is then called core stable if no blocking coalition (of any size) exists. Notice that core stability is a very strong (and hence very desirable) notion of stability, compared with simpler notions, such as Nash stability (which only precludes divergence by a single agent).

Attractive though it may be from the game theory point of view, the notion of core stability presents some serious drawbacks from the point of view of computational complexity. In particular, deciding if an ASHG admits a core stable outcome is not just NP-hard, but in fact Σ_2^p-complete, that is, complete for the second level of the polynomial hierarchy [45], even if the preferences are symmetric (i.e., the input graph is undirected), has bounded degree, and edge weights are bounded by a constant [39]. Compared to simpler notions of stability, such as Nash stability (which is "only" NP-complete [23]), core stability is therefore highly intractable, and this strongly motivates the search for a better understanding of what the source of this intractability is and for ways to deal with it. The focus of this paper is on using notions of graph structure from parameterized complexity to achieve a more fine-grained understanding of the complexity of this problem. Throughout the paper we will concentrate on the case where agent preferences are *symmetric*, that is, the given graphs are undirected. Since most of our results are negative, this (natural) restriction only renders them stronger.

	CSV	CS
$\texttt{vc}$	coNP-complete ($\texttt{vc} = 2$) $(\texttt{vc}w_{\max})^{O(\texttt{vc})}\Delta^2 + O(\texttt{vc}n)$	Σ_2^p-complete ($\texttt{vc} = 12$)
$\texttt{vi}$	coW[1]-hard ($w_{\max} = n^{O(1)}$) $f(\texttt{vi} + w_{\max})n^{O(1)}$	
$\texttt{td}$	coW[1]-hard ($w \in \{-1, 1\}$)	
$\texttt{pw}$		$\nexists 2^{2^{o(\texttt{pw})}} n^{O(1)}$ ($\Delta = O(1)$)
$\texttt{tw}$	$(\Delta w_{\max})^{O(\texttt{tw})} n^{O(1)}$ $2^{O(\texttt{tw}\Delta)}(n + \log w_{\max})^{O(1)}$	$2^{2^{O(\Delta\texttt{tw})}} n^{O(1)}$

Figure 1 The complexity of Core Stability (CS) and Core Stability Verification (CSV) with respect to graph parameters: treewidth ($\texttt{tw}$), pathwidth ($\texttt{pw}$), tree-depth ($\texttt{td}$), vertex integrity ($\texttt{vi}$), and vertex cover ($\texttt{vc}$). We denote by $w_{\max}$ the maximum absolute weight. The hardness results are colored in red and the algorithmic results are colored in blue. The connection between the upper parameter p and the lower parameter q indicates that $q \leq p + 1$ holds for any graph G.

Our results. In this paper we present several results that improve and clarify the state of the art on the complexity of finding core stable outcomes in ASHGs (see Figure 1). We study two closely related problems: Core Stability (CS) and Core Stability Verification (CSV), which correspond to deciding if a core stable partition exists and deciding if a given partition is indeed core stable respectively. Intuitively, the reason CS is complete for the second level of the polynomial hierarchy (and not just NP-complete) is that CSV is also known to be intractable (coNP-complete [16, 43]). Our high-level aim is to understand which parts of the combinatorial structure of the input are responsible for the complexity of these two problems. In order to quantify the input structure we will use standard structural tools from the toolbox of parameterized complexity, such as the notions of treewidth and related parameters[1].

We begin our investigation with CSV and ask the question which restrictions on the input are likely to render the problem tractable (or conversely, what are the sources of the problem's intractability). We identify two possible culprits: the problem could become easy if we either impose restrictions on the graph structure, for example by requiring that the input be of low treewidth or degree, or if we impose restrictions on the allowed edge weights. Our results indicate that these two sources of intractability interact in non-trivial ways: placing restrictions of one type is typically not enough to render the problem tractable, but the problem does sometimes become tractable if we restrict both the graph structure and the allowed weights. More precisely, we show that:

- If we place absolutely no restrictions on the allowed weights, CSV remains hard even on severely restricted instances, that is, graphs of vertex cover 2 (Theorem 2). We find this rather surprising, as this class of graphs (which are essentially stars with one additional vertex) is rarely general enough to render problems intractable.
- One may be tempted to interpret the previous result as an artifact of the exponentially large weights we allow in the input. However, we show that even if we place the restriction that weights are polynomially bounded in the input size, CSV still remains quite hard from

[1] Throughout the paper we assume the reader is familiar with the basics of parameterized complexity, as given for example in [17].

the parameterized perspective, and more precisely coW[1]-hard parameterized by vertex integrity (Theorem 3). Note that graphs with small vertex integrity are graphs where there exists a small separator whose removal breaks down the graph into components of bounded size, so this parameterization is again rather restrictive because it usually easily renders most graph problems almost as tractable as parameterizing by vertex cover [25, 33].

- Finally, we show that even if we insist on weights only being selected from the set $\{-1, 1\}$, CSV is coW[1]-hard parameterized by tree-depth (Theorem 4).

Taken together these results show that CSV is an unusually intractable problem where hardness comes from a combination of two factors: the complexity of dealing with the edge weights and the complexity of dealing with the graph-theoretic structure of the input. We complement the above with several algorithms that paint a clearer picture of the complexity of CSV showing that: (i) CSV is polynomial-time solvable on trees (Theorem 5), hence Theorem 2 cannot be extended to graphs of vertex cover 1 (i.e., stars) (ii) CSV is FPT parameterized by vertex integrity plus the maximum edge weight (Theorem 7), so the hardness result of Theorem 4 cannot be extended to vertex integrity (iii) Theorem 4 is matched by an XP algorithm parameterized by treewidth with parameter dependence $(\Delta w_{\max})^{O(\mathtt{tw})}$, that is, an XP algorithm when weights are polynomially bounded (Theorem 8) (iv) the former algorithm can be improved to an FPT running time (even for unbounded weights) if we parameterize by $\mathtt{tw}+\Delta$ (this was already observed by Peters [38], who gave an algorithm with parameter dependence $2^{O(\Delta^2\mathtt{tw})}$, but we improve this complexity to $2^{O(\Delta\mathtt{tw})}$ in Theorem 9).

The results above paint a comprehensive and rather negative picture on the complexity of CSV, which seems to imply that our main problem, that is, *finding* core-stable partitions, is likely to be even more intractable. We confirm this intuition by showing that CS remains Σ_2^p-complete even on graphs of bounded vertex cover (Theorem 10). One encouraging piece of news, however, is that we did manage to obtain an FPT algorithm when CSV is parameterized by $\mathtt{tw} + \Delta$, so this seems like a case worth considering for CS. Indeed, Peters [38] already showed that CS is FPT for this parameterization, without, however, giving an explicit algorithm (his argument was based on Courcelle's theorem). We improve upon this by giving an explicit algorithm whose dependence is *double-exponential* on $\mathtt{tw} + \Delta$, using the technique of reducing to ∃∀-SAT advocated in [31] (Theorem 11). Despite fixed-parameter tractability, it is fair to say that the running time of our algorithm is quite disappointing. Our main contribution in this part is to show that this is, unfortunately, likely to be optimal: even for instances of bounded degree, the existence of an algorithm with better than double-exponential dependence on treewidth would violate the ETH (Theorem 16). This shows another aspect where core-stability is significantly harder than Nash stability, which has "just" slightly super-exponential dependence in $\mathtt{tw} + \Delta$ [28]. Note that the phenomenon that problems complete for the second level of the polynomial hierarchy tend to have double-exponential complexity in treewidth has been observed before [22, 32, 34, 8]. Along the way, we provide a fine-grained analysis of the complexity of solving ∃∀-SAT parameterized by treewidth, which may be of independent interest.

Finally, we conclude our paper by considering one last relevant parameter: the size of the allowed blocking coalition. We say that a partition is k-core stable if no blocking coalition of size at most k exists. The concept of k-core stability was first proposed in [20] for handling more practical scenarios. For small values of k this is a natural variation of the problem, which could potentially render it more tractable – indeed, for k fixed, CSV is trivially in P and CS is trivially in NP. Unfortunately, we show that not much more is gained from these parameterizations: CSV is coW[1]-hard parameterized by k (even on unweighted graphs); while k-CS is NP-complete for all fixed $k \geq 3$, even on graphs of bounded maximum degree and with bounded weights.

The full version of this paper is available on arXiv, where all missing proofs can be found.

2 Preliminaries

We use standard graph-theoretic notation and focus on undirected graphs. An Additively Separable Hedonic Game (ASHG) is represented by a directed graph $G = (V, E)$, where vertices of V represent the agents, and a weight function $w : E \to \mathbb{Z}$. To simplify notation we will extend w to all pairs of vertices and assume that $w(uv) = 0$ whenever $uv \notin E$. We also assume that the self-utility of an agent is 0, that is, $w(uu) = 0$. If $w(uv) = w(vu)$ for every pair of u, v, an ASHG is called *symmetric* and the input graph is undirected. A partition $\mathcal{P}$ of V is a collection of disjoint subsets of V whose union includes all of V. We will call the sets of such a partition *coalitions*. Slightly abusing notation, we will write, for $u \in V$, $\mathcal{P}(u)$ to denote the set of $\mathcal{P}$ that contains u. The utility of an agent $u \in X$ in a set $X \subseteq V$ is defined as $\texttt{ut}(X, u) = \sum_{v \in X} w(uv)$, while the utility of u in a partition $\mathcal{P}$ is defined as $\texttt{ut}_{\mathcal{P}}(u) = \texttt{ut}(\mathcal{P}(u), u) = \sum_{v \in \mathcal{P}(u)} w(uv)$. Even though we defined w as a function to the integers, we will sometimes allow rational edge weights, but with denominators sufficiently small that it will always be easy to obtain an equivalent integer instance by multiplying all weights by an appropriate integer. We use $w_{\max}$ to denote the maximum *absolute* weight of a given ASHG instance. Unless otherwise stated, we assume that w is given to us encoded in binary (and hence $w_{\max}$ may have value exponential in the input size).

We are chiefly interested in the following notion of stability.

▶ **Definition 1** (Core stability). *A partition $\mathcal{P}$ of an ASHG (G, w) is* core stable*, if there exists no $X \subseteq V(G)$ such that for all $u \in X$ we have $\texttt{ut}(X, u) > \texttt{ut}_{\mathcal{P}}(u)$.*

If the set X mentioned in Definition 1 does exist, then we say that $\mathcal{P}$ is unstable and that X is a *blocking coalition*. For fixed integer values of k, we will also study the notion of k-Core Stability: a partition is k-core stable if no blocking coalition of size at most k exists.

The two computational problems we are interested in are Core Stability (CS) and Core Stability Verification (CSV). In the former problem we are given as input an ASHG and are asked if there exists a core stable partition; in the latter we are also given a specific partition $\mathcal{P}$ and are asked if $\mathcal{P}$ is core stable.

We say that a partition $\mathcal{P}$ of $V(G)$ is *connected* if $G[P]$ is connected for every $P \in \mathcal{P}$. Notice that for both CSV and CS we may assume that the partition $\mathcal{P}$ we seek or we are given is connected, as replacing a disconnected coalition $P \in \mathcal{P}$ with a coalition for each of its components does not change $\texttt{ut}_{\mathcal{P}}(u)$ for any $u \in V$ and hence does not affect stability.

Graph parameters and Parameterized Complexity. We assume the reader is familiar with the basics of parameterized complexity, such as the classes FPT and W[1], as given for example in [17]. We assume that the reader is also familiar with standard structural graph parameters. The parameters we will focus on are treewidth (tw), pathwidth (pw), tree-depth (td), vertex integrity (vi), and vertex cover (vc). For the definitions of treewidth and pathwidth, as well as the corresponding (nice) decompositions we refer the reader to [17]. The vertex integrity $\texttt{vi}(G)$ of a graph G is defined as the minimum k such that there exists a set $S \subseteq V(G)$ (called a $\texttt{vi}(k)$-set) such that the largest component of $G - S$ has order at most $k - |S|$. The tree-depth of a graph G is defined inductively as follows: an isolated vertex

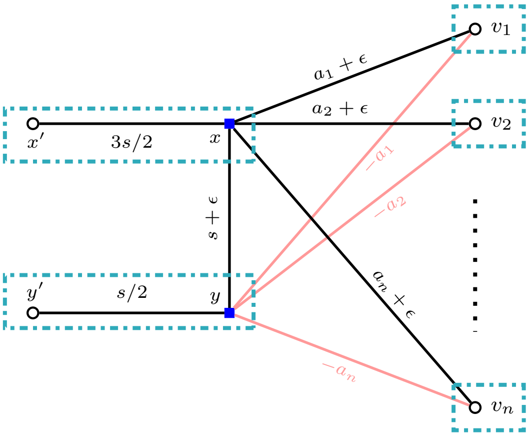


Figure 2 The graph (G, w) constructed in the proof of Theorem 2. The vertex cover is marked by blue squares. The initial partition is given by dot-dashed boxes.

has tree-depth 1; the tree-depth of a disconnected graph is the maximum of the tree-depth of its components; the tree-depth of a connected graph G is defined as $\min_{v\in V(G)} \mathtt{td}(G-v)+1$. The vertex cover of G is the size of the smallest set of vertices of G that intersects all edges.

It is well known that for all graphs G we have $\mathtt{tw}(G) \leq \mathtt{pw}(G) \leq \mathtt{td}(G) \leq \mathtt{vi}(G) \leq \mathtt{vc}(G)+1$ where the second relationship is shown in [9] and the others follow immediately from the definition of the respective parameters. In terms of parameterized complexity these parameters therefore form a hierarchy: if a problem is FPT for a smaller parameter, then it is FPT for the larger ones and conversely if a problem is intractable for a large parameter, then it is intractable for a smaller one. We therefore say that larger parameters are more restrictive, with vertex cover being the most restrictive parameter we consider. We use $\Delta(G)$ to denote the maximum degree of a graph G and omit G when it is clear from the context.

3 Core Stability Verification

In this section we study the complexity of Core Stability Verification (CSV). What we discover is that this is an unusually intractable problem, even for quite restricted parameterizations. We complete the picture by giving complementing algorithms.

3.1 Hardness Results

We first prove the following three hardness results.

▶ **Theorem 2.** *Core Stability Verification is weakly coNP-complete on graphs of vertex cover number 2.*

Proof. First note that CSV is in coNP as it is polynomial to verify that a given coalition is blocking. We give a reduction from Partition. Given a set of positive integers $A = \{a_1, \ldots, a_n\}$, the Partition problem asks whether there exists a subset A' of A such that $\sum_{a\in A'} a = s/2$ where $s = \sum_{a\in A} a$. This problem is well-known to be weakly NP-complete [24].

We construct an instance of CSV. The construction is depicted in Figure 2. First, we create n vertices $v_1, \ldots, v_n$ corresponding to $a_1, \ldots, a_n \in A$ and four vertices x, y, x', y'. Then we add edges v_ix of weight $a_i+\epsilon$, v_iy of weight $-a_i$, xx' of weight $3s/2$, yy' of weight $s/2$, and xy of weight $s+\epsilon$. Here, without loss of generality, let ϵ be an integer sufficiently smaller than $\min_i a_i$; this can be achieved for example by multiplying all elements of A (and s) by n, and setting $\epsilon = 1$. Let (G, w) be the constructed graph. The partition $\mathcal{P}$ to verify consists of $\{x, x'\}$, $\{y, y'\}$ and singletons $\{v_1\}, \ldots, \{v_n\}$. Also note that $\{x, y\}$ is a vertex cover of G.

If there exists $A' \subseteq A$ such that $\sum_{a\in A'} a = s/2$, then the coalition $X = \{v_i : a_i \in A'\} \cup \{x, y\}$ blocks $\mathcal{P}$. To see this, observe that the utility of each vertex in X increases by at least ϵ and thus X is a blocking coalition of $\mathcal{P}$.

Conversely, suppose that there exists a blocking coalition X of $\mathcal{P}$. Clearly, X contains neither x' nor y'. If X does not contain x, no vertex can have positive utility in X. Thus, they also do not join X as they have non-negative utility in $\mathcal{P}$. Thus, X must contain x. To increase the utility of x, y must be contained in X. In particular, if y was not contained in X, then the utility of x would be at most $s + n\epsilon < 3s/2$. Since y must have utility more than $s/2$ in X, it holds that $\sum_{v_i\in X} a_i \le s/2$. Finally, since x obtains at least $s+\epsilon$ utility in X from $y \in X$, and x has $3/2s$ utility in P, we have that $s+\epsilon+\sum_{v_i\in X} a_i > 3/2s$, that is $\sum_{v_i\in a_i} > 1/2s - \epsilon$. Since ϵ is sufficiently smaller than $\min_i a_i$, this implies that $\sum_{v_i\in X} a_i = s/2$ and hence there exists a subset A' of A such that $\sum_{a\in A'} a = s/2$. ◀

We use a similar but more involved construction to reduce Bin Packing to CSV.

▶ **Theorem 3.** *Core Stability Verification is coW[1]-hard parameterized by vertex integrity, even if all weights are bounded by a polynomial in the input size.*

We prove the third hardness result by a reduction from Bounded Degree Deletion.

▶ **Theorem 4.** *Core Stability Verification is coW[1]-hard parameterized by tree-depth, even if all weights are in $\{-1, 1\}$.*

3.2 Algorithms

In this section, we prove the algorithmic results complementing the hardness results of the previous section.

▶ **Theorem 5.** *Core Stability Verification is polynomial time solvable on trees.*

▶ **Theorem 6.** *Core Stability Verification can be solved in time $(\mathtt{vc}w_{\max})^{O(\mathtt{vc})}\Delta^2 + O(\mathtt{vc}n)$.*

Proof. Given a graph (G, w) and a partition $\mathcal{P}$ of $V(G)$, we check whether there is a blocking coalition $X \subseteq V(G)$ of $\mathcal{P}$. In the algorithm, we first compute a minimum vertex cover S of size $\mathtt{vc}$ in time $O(1.25284^{\mathtt{vc}} + \mathtt{vc}n)$ [29]. Then we guess the intersection S' of X and S. The number of possible candidates of the intersection S' is at most $2^{|S|}$. Let $I' \subseteq V(G)\setminus S$ be the set of vertices in $V(G)\setminus S$ such that each vertex $u \in I'$ satisfies $\sum_{v\in N_G(u)\cap S'} w(uv) > \mathtt{ut}_{\mathcal{P}}(u)$. The vertices in I' could form a blocking coalition of $\mathcal{P}$ by cooperating with the vertices in S'. In order for X to become a blocking coalition, all the vertices in S' must have larger utility in X than their utility in $\mathcal{P}$ after some vertices in I' joined X. This condition can be represented as an Integer Linear Program (ILP) as follows:

$$\sum_{v\in I'} w(uv)x_v + \sum_{v\in N_G(u)\cap S'} w(uv) \ge \mathtt{ut}_{\mathcal{P}}(u) + 1 \quad \forall u \in S' \tag{1}$$

$$x_v \in \{0, 1\} \quad \forall v \in I' \tag{2}$$

where the variable x_v represents whether vertex $v \in I'$ joins X. On the left hand side of (1), $\sum_{v \in N_G(u) \cap S'} w(uv)$ represents the contribution of edge weights in S' to the utility of u. Clearly, the ILP is feasible if and only if there is a blocking coalition X because the utility of each agent in X strictly increases. Note that we suppose that each edge weight is an integer.

Here, the feasibility check of an ILP $\{Ax = b, x \geq 0, x \in \mathbb{Z}^n\}$ can be solved in time $(m \cdot \Lambda)^{O(m)} \cdot ||b||_\infty^2$ where $A \in \mathbb{Z}^{n \times m}$, $b \in \mathbb{Z}^m$, and Λ is an upper bound on each absolute value of an entry in A [19]. By adding a slack variables in $\mathbb{Z}$ to each inequality in (1), the ILP can be transformed into the form $\{Ax = b, x \geq 0, x \in \mathbb{Z}^n\}$ where $n = |I'| + |S'|$ and $m = |S'|$. Since the maximum absolute value of coefficients of variables is $w_{\max}$, the range of $\mathtt{ut}_{\mathcal{P}}(u)$ and $\sum_{v \in N_G(u) \cap S'} w(uv)$ is $[-w_{\max}\Delta, w_{\max}\Delta]$, the ILP can be solved in time $(\mathtt{vc}w_{\max})^{O(\mathtt{vc})}(w_{\max}\Delta)^2 = (\mathtt{vc}w_{\max})^{O(\mathtt{vc})}\Delta^2$. Thus, the total running time is $O(1.25284^{\mathtt{vc}} + \mathtt{vc}n) + 2^{\mathtt{vc}}(\mathtt{vc}w_{\max})^{O(\mathtt{vc})}\Delta^2 = (\mathtt{vc}w_{\max})^{O(\mathtt{vc})}\Delta^2 + O(\mathtt{vc}n)$. ◀

▶ **Theorem 7.** *Core Stability Verification is in FPT parameterized by* $\mathtt{vi} + w_{\max}$.

▶ **Theorem 8.** *Core Stability Verification can be computed in time* $(\Delta w_{\max})^{O(\mathtt{tw})} n^{O(1)}$.

▶ **Theorem 9.** *Core Stability Verification can be computed in time* $2^{O(\mathtt{tw}\Delta)}(n + \log w_{\max})^{O(1)}$.

4 Core Stability

In this section we study the complexity of Core Stability (CS). We first show that CS remains Σ_2^p-complete even on graphs of bounded vertex cover number (Theorem 10).

The second part of this section is dedicated to extending our understanding of the complexity of CS parameterized by $\mathtt{tw} + \Delta$. We give an algorithm for CS running in time $2^{2^{O(\Delta\mathtt{tw})}} n$ (Theorem 11) improving on the previous algorithm based on Courcelle's Theorem by Peters [38]. In order to avoid having to formulate a tedious dynamic programming algorithm, we instead obtain our algorithm via a reduction to ∃∀-SAT, which is known to be solvable in double-exponential time (in treewidth) [15]. We complement these results by giving an ETH based lower bound of $2^{2^{o(\mathtt{pw})}}$ on graphs of bounded degree (Theorem 16). This shows that the double-exponential dependence of our algorithm on treewidth is in fact inevitable, and confirms a pattern shown by other Σ_2^p-complete problems [32].

4.1 Core stability on graphs of bounded vertex cover number

In this section we prove the following result.

▶ **Theorem 10.** *Core Stability is* Σ_2^p*-complete on graphs of vertex cover number 12.*

To obtain this result we use a variation of the constructions used to prove Theorem 2 and Theorem 3; however reducing from an appropriate variant of Partition namely ∃∀-partition. To control how core stable partitions behave we use a gadget utilizing the non-core stable graph given in [3]. Details of this auxiliary gadgets construction and behaviour is omitted due to space limitation.

4.2 Core stability parameterized by maximum degree and treewidth

We first prove the algorithmic result of the section.

▶ **Theorem 11.** *Core Stability can be solved in time* $2^{2^{O(\Delta\mathtt{tw})}} n^{O(1)}$.

Before we prove Theorem 11, let us sketch our high-level strategy. Given an instance of Core Stability, we want to produce an equivalent instance ϕ of $\exists\forall$-SAT, such that ϕ has treewidth roughly $\Delta\mathtt{tw}$, where Δ and $\mathtt{tw}$ are the maximum degree and treewidth of the original instance. We could then use the known (double-exponential) algorithm for $\exists\forall$-SAT [15] to solve our problem. Intuitively, we would then attempt to use the existential part of ϕ to encode the "there exists a partition" part of the problem, and the universal part to encode the "all blocking coalitions fail to be blocking" part.

Fundamentally, this strategy is sound and works in a relatively straightforward way for the universal part: we use a boolean variable for each vertex (to encode whether it belongs in the potential blocking coalition) and to check that a coalition fails to be blocking for a vertex v we need to place a constraint on v and all its (at most Δ) neighbors. This means that a tree decomposition of ϕ should be constructible from a tree decomposition of the square of the original graph, which would have width at most $\Delta\mathtt{tw}$.

Where we run into some more difficulties, however, is in encoding the existential part. Intuitively, this is because encoding the partition of the vertices of a bag into coalitions requires a super-linear number of bits, hence it is not sufficient to define a variable for each vertex. Indeed, to simplify things, we define a variable for each *pair* of vertices that appear together in a bag, encoding whether they are together in a coalition. This means that the treewidth of the formula ϕ we construct is in fact not $O(\Delta\mathtt{tw})$, but actually can only be upper-bounded by $O(\mathtt{tw}^2 + \Delta\mathtt{tw})$.

Nevertheless, we insist on obtaining an algorithm that is double-exponential "only" in $\Delta\mathtt{tw}$, and not in $\mathtt{tw}^2$. In order to circumvent this difficulty we observe that the term that is super-linear in treewidth only depends on existentially quantified variables. Thankfully, we manage to show, via an argument that is more careful than that of [15], that $\exists\forall$-SAT has a time complexity that only needs to be double-exponential in the number of *universally quantified* variables of each bag (Proposition 12). Using this, we are able to show that the second exponent of the running time is "only" $O(\Delta\mathtt{tw})$, which as we show later is optimal, even when $\Delta = O(1)$, under the ETH.

Let us now give some more details. We first recall that $\exists\forall$-SAT is a variant of the SAT problem where we aim to decide the satisfiability of a given quantified Boolean formula (QBF) ϕ which is of the form $\exists x_1 \dots \exists x_k \forall y_1 \dots \forall y_\ell \psi$ where ψ is a DNF formula on variables $x_1, \dots, x_k, y_1, \dots, y_\ell$. Two common ways of associating structure of satisfiability problems is to consider the primal or incidence graph of the formula. The primal graph of a formula ϕ (in CNF or DNF) is a graph on the set of variables of ϕ where two variables are adjacent if they appear in the same clause. Similarly, the incidence graph is a bipartite graph on the set of variables and clauses of ϕ where a variable is adjacent to all clauses it appears in. For convenience, we use a variant of $\exists\forall$-SAT. We say that a QBF ϕ is in $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$ if ϕ can be written as

$$\phi = \exists x_1 \dots \exists x_k \bigwedge_{i=1}^{k'} d_i \quad \forall y_1 \dots \forall y_\ell \bigvee_{i=1}^{\ell'} c_i$$

for some $k, k', \ell, \ell' \in \mathbb{N}$ where d_i are disjunctive clause over variables $x_1, \dots, x_k$ containing at most 3 literals per clause and c_i are conjunctive clauses over variables $x_1, \dots, x_k, y_1, \dots, y_\ell$. We present an algorithm for $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$ in several steps. First, we give an algorithm with an improved running time than that of [15].

▶ **Proposition 12.** *There is an algorithm that takes as input an instance of* $\exists\forall$-SAT $\exists x \forall y \phi(x, y)$*, where* x, y *are tuples of boolean variables,* ϕ *is in* 3-DNF*, and a tree decomposition of the primal graph of* ϕ *where each bag contains at most* $t_\exists$ *existentially quantified variables and at most* $t_\forall$ *universally quantified variables and decides if the input is satisfiable in time* $2^{O(t_\exists + 2^{t_\forall})}|\phi|^{O(1)}$.

▶ **Proposition 13.** *There is an algorithm that takes as input an* $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$-SAT *instance* ϕ *and a tree decomposition of its incidence graph of width* t *that contains at most* $t_\forall$ *universally quantified variables in each bag and at most* 2 *clauses in each bag, and decides* ϕ *in time* $2^{O(t2^{t_\forall})}|\phi|$.

Proof of Theorem 11. We prove Theorem 11 by a reduction to $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$-SAT. Let (G, w) be an instance of CS and (T, β) be a rooted tree decomposition of G. Here, we denote the bag of a node $t \in T$ by $\beta(t)$. First we let (G', w') be the graph obtained from (G, w) by adding edges uv of weight 0 for every pair of vertices u, v appearing in a bag together. It is straightforward to see that (G, w) is a YES-instance of CS if and only if (G', w') is a YES-instance of CS. Additionally, G' is chordal and (T, β) is a tree decomposition of G'.

We construct an instance ϕ of $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$-SAT. We introduce a variable x_e for every $e \in E(G')$ and a variable y_u for every vertex $u \in V(G')$. An assignment $\alpha_X : \{x_e : e \in E(G')\} \to \{0, 1\}$ represents a subset of $E(G')$ and an assignment $\alpha_Y : \{y_u : u \in V(G')\} \to \{0, 1\}$ represents a subset of $V(G')$. Intuitively, a partition of $V(G')$ corresponds to a set of edges, i.e., by including all edges that are incident to vertices from the same part. For every $t \in V(T)$ we define a formula ϕ_t essentially expressing transitivity. We use formula ϕ_t to enforce sufficient criteria for the set of edges represented by an assignment $\alpha_X : \{x_e : e \in E(G')\} \to \{0, 1\}$ to correspond to a partition of $V(G')$. Let

$$\phi_t = \bigwedge_{u_1,u_2,u_3 \in \beta(t)} \big(x_{u_1u_2} \wedge x_{u_2u_3}\big) \to x_{u_1u_3} = \bigwedge_{u_1,u_2,u_3 \in \beta(t)} \Big(\neg x_{u_1u_2} \vee \neg x_{u_2u_3} \vee x_{u_1u_3}\Big).$$

For a fixed partition $\mathcal{P}$ represented by some assignment $\alpha_X : \{x_e : e \in E(G')\} \to \{0, 1\}$ our formula needs to ensure that no blocking coalitions exist. This is realized by guaranteeing that for each assignment $\alpha_Y : \{y_u : u \in V(G')\} \to \{0, 1\}$ the set corresponding to α_Y is not blocking. Intuitively, the formula ϕ_u defined below ensures that the utility of vertex u in $\mathcal{P}$ is at least as large as the utility of u in the coalition represented by α_Y. To realize this, we make sure that the set N of neighbors of u which are in the coalition represented by α_Y and the set $\tilde{N}$ of neighbors of u which are in the same part as u in $\mathcal{P}$ satisfy $\sum_{v \in N} w(uv) \leq \sum_{v \in \tilde{N}} w(uv)$. For $u \in V(G')$ let

$$\phi_u = \bigvee_{\substack{N, \tilde{N} \subseteq N_{G'}(u), \\ \sum_{v \in N} w(uv) \leq \sum_{v \in \tilde{N}} w(uv)}} \Bigg(y_u \wedge \bigwedge_{v \in N} y_v \wedge \bigwedge_{v \in N_{G'}(u) \setminus N} \neg y_v \wedge \bigwedge_{v \in \tilde{N}} x_{uv} \wedge \bigwedge_{v \in N_{G'}(u) \setminus \tilde{N}} \neg x_{uv}\Bigg).$$

We now define the formula ϕ to be

$$\phi = \exists x_{e_1} \dots \exists x_{e_m} \bigwedge_{t \in V(T)} \phi_t \quad \forall y_{v_1} \dots \forall y_{v_n} \big(\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n}\big) \vee \bigvee_{u \in V(G')} \phi_u.$$

Observe that ϕ is in $\exists 3\,\mathrm{CNF}\,\forall\,\mathrm{DNF}$. In the following we prove that (G', w') is a YES-instance of CS if and only if ϕ is satisfiable.

First assume that $\mathcal{P}$ is a core stable partition of $V(G')$. We define an assignment $\alpha_X : \{x_e : e \in E(G')\} \to \{0, 1\}$ by setting $\alpha_X(x_e) = 1$ if e is incident to two vertices residing in the same part of $\mathcal{P}$ and $\alpha_X(x_e) = 0$ otherwise. This assignment satisfies ϕ_t for every $t \in V(T)$ as for any three vertices $u_1, u_2, u_3 \in \beta(t)$ it holds that if $\alpha_X(x_{u_1u_2}) = 1$ and $\alpha_X(x_{u_2u_3}) = 1$, then u_1, u_2 and u_3 must reside in the same part of $\mathcal{P}$ and hence $\alpha_X(x_{u_1u_3}) = 1$. Furthermore, consider any assignment $\alpha_Y : \{y_u : u \in V(G')\} \to \{0, 1\}$ and let $X = \{u \in V(G') : \alpha_Y(y_u) = 1\}$. We have to argue that the formula $\big(\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n}\big) \vee \bigvee_{u \in V(G')} \phi_u$ is satisfied under the assignments α_X and α_Y. In case that $X = \emptyset$, the clause $(\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n})$ is satisfied. On the other hand, if $X \neq \emptyset$, then there must be a vertex $u \in X$ whose utility in $\mathcal{P}$ is at least as large as its utility in X as the set X cannot be a blocking coalition. Hence, for the sets $N = N_{G'}(u) \cap X$ and $\tilde{N} = N_{G'}(u) \cap P$, where $P \in \mathcal{P}$ is the part containing u, we have that $\sum_{v \in N} w(uv) \leq \sum_{v \in \tilde{N}} w(uv)$. Therefore, ϕ_u contains the clause $(y_u \wedge \bigwedge_{v \in N} y_v \wedge \bigwedge_{v \in N_{G'}(u) \setminus N} \neg y_v \wedge \bigwedge_{v \in \tilde{N}} x_{uv} \wedge \bigwedge_{v \in N_{G'}(u) \setminus \tilde{N}} \neg x_{uv})$ and this clause is satisfied under the assignment α_X and α_Y by choice of N and $\tilde{N}$. This shows that ϕ is satisfiable.

On the other hand, assume that ϕ is satisfiable and let $\alpha_X : \{x_e : e \in E(G')\} \to \{0, 1\}$ be an assignment such that $\bigwedge_{t \in V(T)} \phi_t$ as well as $\forall y_{v_1} \ldots \forall y_{v_n} (\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n}) \vee \bigvee_{u \in V(G')} \phi_u$ is satisfied under α_X. We let $E_X = \{e \in E(G') : \alpha_X(x_e) = 1\}$ and define a partition $\mathcal{P}$ by letting every part of $\mathcal{P}$ correspond to the vertices of a connected component of the graph $(V(G'), E_X)$. To show that the partition $\mathcal{P}$ is core stable we use the following claim.

▷ Claim 14. For every edge $uv \in E(G')$ it holds that $uv \in E_X$ if and only if u and v are contained in the same part of $\mathcal{P}$.

Towards a contradiction assume that $\mathcal{P}$ is not core stable and let X be a blocking coalition. We define an assignment $\alpha_Y : \{y_u : u \in V(G')\} \to \{0, 1\}$ by setting $\alpha_Y(y_u) = 1$ if $u \in X$ and $\alpha_Y(y_u) = 0$ otherwise. By assumption, the DNF formula $(\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n}) \vee \bigvee_{u \in V(G')} \phi_u$ is satisfied under assignment α_X and α_Y. As X cannot be empty there is at least one $u \in V(G')$ such that $\alpha_Y(y_u) = 1$ and hence the clause $(\neg y_{v_1} \wedge \cdots \wedge \neg y_{v_n})$ cannot be satisfied under the assignment α_Y which implies that some clause in $\bigvee_{u \in V(G')} \phi_u$ must be satisfied. Let $u \in V(G')$ and $N, \tilde{N} \subseteq N_{G'}(u)$ with $\sum_{v \in N} w(uv) \leq \sum_{v \in \tilde{N}} w(uv)$ such that the clause $(y_u \wedge \bigwedge_{v \in N} y_v \wedge \bigwedge_{v \in N_{G'}(u) \setminus N} \neg y_v \wedge \bigwedge_{v \in \tilde{N}} x_{uv} \wedge \bigwedge_{v \in N_{G'}(u) \setminus \tilde{N}} \neg x_{uv})$ is satisfied under assignments α_X and α_Y. This implies that $N = N_{G'}(u) \cap X$ and $\tilde{N} = \{v \in V(G') : uv \in E_X\}$. Since by Claim 14, $\{v \in V(G') : uv \in E_X\} = N_{G'}(u) \cap P$, where P is the part of $\mathcal{P}$ containing u, this implies that the utility of u in X is less or equal to the utility of u in $\mathcal{P}$. As this contradicts our assumption that X is a blocking coalition it follows that $\mathcal{P}$ is core stable.

▷ Claim 15. The formula ϕ has incidence treewidth at most $\Delta(G')\mathtt{tw}(G') + \mathtt{tw}(G')^2 + 2$. Furthermore, we can construct a decomposition of that width which contains at most 2 clauses per bag and at most $(\Delta(G') + 1)\mathtt{tw}(G')$ universally quantified variables per bag.

As (T, β), G' and the formula ϕ can be computed in time $\mathcal{O}(\mathtt{tw}(G')^2 n)$, combining the reduction with the algorithm from Proposition 13 yields a $2^{2^{\mathcal{O}(\Delta \mathtt{tw})}} n^{O(1)}$ time algorithm. ◀

Finally, we prove our ETH based lower bound.

▶ **Theorem 16.** *Unless the ETH fails, there is no algorithm for* Core stability *running in time $2^{2^{o(\mathtt{pw})}} n^{\mathcal{O}(1)}$ even if G has bounded degree and weights are constant.*

We give an overview of our construction. Given an instance ϕ of $(3,3)$-SAT (each variable appears at most 3 times) we construct a graph (G, w) and show that ϕ is satisfiable if and only if (G, w) is core stable. Firstly, we use a gadget based on the non-core stable graph given in [3] to control potential core stable partitions. The auxiliary gadget is a non-core stable graph H that we attach with positive weight edges at a set S of vertices. Because H is not core stable any partition must place some vertex of H in a part with some $s \in S$. Using negative weight edges we can control this behaviour even further. We achieve that $\{h, s\}$ must be in any core stable partition for one vertex $s \in S$ where $h \in V(H)$ is a fixed vertex of H.

We now describe the construction. Every variable x_i, $i \in [n]$ of ϕ is represented by two vertices y_i and $\neg y_i$ and for each $i \in [n]$ we attach one auxiliary gadget at $\{y_i, \neg y_i\}$. We add further vertices (details follow in the next paragraph). Each of these additional vertices v either has its private auxiliary gadget attached at $\{v\}$ or is not connected to any auxiliary gadgets. We define a special set of partitions of $V(G)$ which we refer to as candidate partitions as follows. Any candidate partition $\mathcal{P}$ has to contain either $\{h, y_i\}$ or $\{h, \neg y_i\}$ (but not both) where h is a vertex of the respective auxiliary gadget. For any other vertex v without an attached auxiliary gadget (excluding vertices of auxiliary gadgets) any candidate parition $\mathcal{P}$ has to contain $\{v\}$. For any vertex v with an auxiliary gadget attached at $\{v\}$ any candidate partition $\mathcal{P}$ has to contain the set $\{h, v\}$ where h is a vertex of respective auxiliary gadget. Using the properties of the auxiliary gadget, we can argue that any core stable partition has to be a candidate partition. By the construction, there is a correspondence between assignments and candidate partitions, i.e., $\alpha(x_i) = 1$ if and only if $\{h, y_i\} \in \mathcal{P}$ for candidate partition $\mathcal{P}$ and the corresponding assignment α.

Any blocking coalition of a candidate partition allows us to find a clause which is not satisfied under the assignment which corresponds to the candidate partition and vice versa. This is realized as follows. We take $2m + 1$ cycles $U^1, \ldots, U^m, V^1, \ldots, V^m, Z$ of length $3n$ where $2^m \leq 3n$ is the number of clauses of ϕ. By choosing suitable edge weights, we enforce that any blocking coalition of any candidate partition contains Z and either U^k or V^k (but not both) for every $k \in [m]$. For now, we call any set containing Z, U^k or V^k (but not both) for every $k \in [m]$ (and some other vertices we neglect here) a candidate blocking coalition. We number the clauses of ϕ in such a way that each candidate blocking coalition corresponds to a clause. More sprecifically, the j-th clause corresponds to the candidate blocking coalition X in which $U^k \in X$ if and only if the k-th bit of j in binary is 0.

Each vertex in Z corresponds to the appearance of a variable. Assume $z \in Z$ corresponds to the appearance of variable x_i in clause c_j. We connect z to either y_i or $\neg y_i$ dependent on whether x_i appears negated in c_j. We connect z to either U^k or V^k for every $k \in [m]$ dependent on whether the k-th bit of j in binary is 0 or 1 using a gadget we call clause selection gadget. The gadget enforces that z obtains a $+1$ towards its total utility in X if and only if the candidate blocking coalition X does not encode the clause c_j. By choice of edge weights, we ensure that vertex z can only be convinced to join a blocking coalition if it either gets utility $+1$ from its clause selection gadget or utility $+1$ from y_i (or $\neg y_i$, resp.). On the other hand, y_i ($\neg y_i$, resp.) can only be convinced to join a blocking coalition if it appears as a singleton in the partition we are trying to block and hence the corresponding literal is false. In conclusion, for any candidate partition $\mathcal{P}$ there is a blocking coalition X if and only if for the clause corresponding to X each literal is false. Hence, (G, w) is core stable if and only if ϕ is satisfiable.

5 k-Core Stability

In this section we consider the complexity of finding and verifying k-core stable partitions, when the size of the allowed blocking coalitions k is a parameter. Even though the two problems do become easier when k is a fixed constant (because we can check all possible blocking coalitions in polynomial time), we show that it is likely that not much more can be gained from this assumption: k-CSV is coW[1]-hard parameterized by k (Theorem 17), while k-CS is NP-complete even if $k \geq 3$ is a fixed constant (Theorem 19). On the positive side, we do show that finding 2-core stable partitions is in P, but it is worth noting that the fact that we consider undirected graphs is crucial to obtain even this small tractable case.

▶ **Theorem 17.** *k-Core Stability Verification is in XP when parameterized by k whereas coW[1]-hard even on unweighted graphs.*

Proof. The upper bound can be easily shown by brute force. That is, given a coalition structure $\mathcal{P}$, for each coalition X of size at most k, we check if each agent v in X has higher utility than in $\mathcal{P}$. The running time of brute force is $n^{O(k)}$.

Then we show that k-CSV is W[1]-hard even on unweighted graphs. We give a reduction from k-Clique. Given a graph G, we attach $k-2$ pendant vertices for each vertex in $V(G)$. Let P_v be the set of pendant vertices for v. We set $\mathcal{P} = \{P_v \cup \{v\} : v \in V(G)\}$ as a coalition structure to verify.

In the following, we show that there exists a k-clique in G if and only if there exists a blocking coalition for $\mathcal{P}$. Let C be a clique of size k in G. For C, each vertex in C has the utility $k-1$. Since $v \in V$ has the utility $k-2$ in $\mathcal{P}$, C is a blocking coalition for $\mathcal{P}$. Conversely, let X be a blocking coalition for $\mathcal{P}$. Since vertices in $\bigcup_{v\in V(G)} P_v$ have maximum utility 1 in $\mathcal{P}$, they do not join X. Thus, X is a subset of $V(G)$. Since the utility of $v \in V(G)$ is $k-2$ in $\mathcal{P}$ and $|X| = k$, X is a clique of size k. ◀

▶ **Theorem 18.** *Every graph admits a 2-core stable partition and 2-Core Stability can be solved in polynomial time.*

Proof. Given a weighted graph (G, w), start with the partition where every vertex is a singleton. Order the positive-weight edges in non-increasing order $e_1, e_2, \ldots, e_m$. For each e_i, do the following: if the endpoint of e_i are currently singletons, merge them into a cluster of size 2; otherwise move to the next edge. The resulting partition $\mathcal{P}$ is 2-core stable because if there was a blocking coalition of size 2, it would have to induce an edge $e_i = uv$. However, when e_i is considered, at least one of u, v was not a singleton. Therefore, the utility of that vertex must be larger in $\mathcal{P}$ than in the coalition $\{u, v\}$ contradicting the assumption that $\{u, v\}$ is a blocking coalition. ◀

▶ **Theorem 19.** *For any fixed $k \geq 3$, k-Core Stability is NP-complete on bounded degree graphs, even if the weights are constant.*

6 Conclusion

The general tenor of our results indicates that core stability is an algorithmically highly intractable notion: even for very restricted input structures, obtaining efficient algorithms seems out of reach; and even for the few cases where positive fixed-parameter tractability results can be obtained, complexity lower bounds still push the parameter dependence to prohibitive levels. Despite the above, we believe that a promising avenue for future research may be the further investigation of k-core stability. Even though we have shown that parameterizing the problem by k alone does not help, it would be interesting to ask whether parameterizing at the same time by both k and a structural parameter (such as treewidth) could help us evade the lower bounds that apply to each case individually. Finally, investigating the parameterized complexity of core stability in other variants of hedonic games such as fractional hedonic games [2, 26, 20] is another promising direction.

References

1 Alessandro Aloisio, Michele Flammini, and Cosimo Vinci. The impact of selfishness in hypergraph hedonic games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1766–1773. AAAI Press, 2020. doi:10.1609/AAAI.V34I02.5542.

2 Haris Aziz, Florian Brandl, Felix Brandt, Paul Harrenstein, Martin Olsen, and Dominik Peters. Fractional hedonic games. *ACM Trans. Economics and Comput.*, 7(2):6:1–6:29, 2019. doi:10.1145/3327970.

3 Haris Aziz, Felix Brandt, and Hans Georg Seedig. Computing desirable partitions in additively separable hedonic games. *Artif. Intell.*, 195:316–334, 2013. doi:10.1016/j.artint.2012.09.006.

4 Haris Aziz and Rahul Savani. Hedonic games. In *Handbook of Computational Social Choice*, pages 356–376. Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.016.

5 Coralio Ballester. NP-completeness in hedonic games. *Games Econ. Behav.*, 49(1):1–30, 2004. doi:10.1016/J.GEB.2003.10.003.

6 Nathanaël Barrot, Kazunori Ota, Yuko Sakurai, and Makoto Yokoo. Unknown agents in friends oriented hedonic games: Stability and complexity. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1756–1763. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33011756.

7 Nathanaël Barrot and Makoto Yokoo. Stable and envy-free partitions in hedonic games. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 67–73. ijcai.org, 2019. doi:10.24963/ijcai.2019/10.

8 Ivan Bliznets and Markus Hecher. Tight double exponential lower bounds. In Xujin Chen and Bo Li, editors, *Theory and Applications of Models of Computation - 18th Annual Conference, TAMC 2024, Hong Kong, China, May 13-15, 2024, Proceedings*, volume 14637 of *Lecture Notes in Computer Science*, pages 124–136. Springer, 2024. doi:10.1007/978-981-97-2340-9_11.

9 Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. doi:10.1006/JAGM.1995.1009.

10 Niclas Boehmer and Edith Elkind. Individual-based stability in hedonic diversity games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1822–1829. AAAI Press, 2020. doi:10.1609/AAAI.V34I02.5549.

11 Anna Bogomolnaia and Matthew O Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002. doi:10.1006/GAME.2001.0877.

12 Felix Brandt, Martin Bullinger, and Anaëlle Wilczynski. Reaching individually stable coalition structures. *ACM Trans. Economics and Comput.*, 11:4:1–4:65, 2023. doi:10.1145/3588753.

13 Martin Bullinger and Stefan Kober. Loyalty in cardinal hedonic games. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 66–72. ijcai.org, 2021. doi:10.24963/ijcai.2021/10.

14 Katarína Cechlárová. Stable partition problem. In *Encyclopedia of Algorithms*, pages 2075–2078. Springer, 2016. doi:10.1007/978-1-4939-2864-4_397.

15 Hubie Chen. Quantified constraint satisfaction and bounded treewidth. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 161–165. IOS Press, 2004.

16 Jiehua Chen, Gergely Csáji, Sanjukta Roy, and Sofia Simola. Hedonic games with friends, enemies, and neutrals: Resolving open questions and fine-grained complexity. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 251–259. ACM, 2023. doi:10.5555/3545946.3598644.

17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Andreas Darmann, Edith Elkind, Sascha Kurz, Jérôme Lang, Joachim Schauer, and Gerhard J. Woeginger. Group activity selection problem with approval preferences. *Int. J. Game Theory*, 47(3):767–796, 2018. doi:10.1007/s00182-017-0596-4.

19 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

20 Angelo Fanelli, Gianpiero Monaco, and Luca Moscardelli. Relaxed core stability in fractional hedonic games. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 182–188. ijcai.org, 2021. doi:10.24963/ijcai.2021/26.

21 Michele Flammini, Bojana Kodric, Gianpiero Monaco, and Qiang Zhang. Strategyproof mechanisms for additively separable and fractional hedonic games. *J. Artif. Intell. Res.*, 70:1253–1279, 2021. doi:10.1613/JAIR.1.12107.

22 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Tight (double) exponential bounds for NP-complete problems: Treewidth and vertex cover parameterizations. *CoRR*, abs/2307.08149, 2023. doi:10.48550/arXiv.2307.08149.

23 Martin Gairing and Rahul Savani. Computing stable outcomes in symmetric additively separable hedonic games. *Math. Oper. Res.*, 44(3):1101–1121, 2019. doi:10.1287/MOOR.2018.0960.

24 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

25 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

26 Tesshu Hanaka, Airi Ikeyama, and Hirotaka Ono. Maximizing utilitarian and egalitarian welfare of fractional hedonic games on tree-like graphs. In Weili Wu and Jianxiong Guo, editors, *Combinatorial Optimization and Applications - 17th International Conference, COCOA 2023, Hawaii, HI, USA, December 15-17, 2023, Proceedings, Part I*, volume 14461 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2023. doi:10.1007/978-3-031-49611-0_28.

27 Tesshu Hanaka, Hironori Kiya, Yasuhide Maei, and Hirotaka Ono. Computational complexity of hedonic games on sparse graphs. In *PRIMA*, volume 11873 of *Lecture Notes in Computer Science*, pages 576–584. Springer, 2019. doi:10.1007/978-3-030-33792-6_43.

28 Tesshu Hanaka and Michael Lampis. Hedonic games and treewidth revisited. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 64:1–64:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.64.

29 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.40.

30 Ayumi Igarashi, Kazunori Ota, Yuko Sakurai, and Makoto Yokoo. Robustness against agent failure in hedonic games. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 364–370. ijcai.org, 2019. doi:10.24963/ijcai.2019/52.

31 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to Courcelle's theorem. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018. doi:10.1007/978-3-319-94144-8_15.

32 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

33 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.34.

34 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.28.

35 Kazunori Ohta, Nathanaël Barrot, Anisse Ismaili, Yuko Sakurai, and Makoto Yokoo. Core stability in hedonic games among friends and enemies: Impact of neutrals. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 359–365. ijcai.org, 2017. doi:10.24963/ijcai.2017/51.

36 Martin Olsen. Nash stability in additively separable hedonic games and community structures. *Theory Comput. Syst.*, 45(4):917–925, 2009. doi:10.1007/S00224-009-9176-8.

37 Martin Olsen, Lars Bækgaard, and Torben Tambo. On non-trivial Nash stable partitions in additive hedonic games with symmetric 0/1-utilities. *Inf. Process. Lett.*, 112(23):903–907, 2012. doi:10.1016/J.IPL.2012.08.016.

38 Dominik Peters. Graphical hedonic games of bounded treewidth. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 586–593. AAAI Press, 2016. doi:10.1609/AAAI.V30I1.10046.

39 Dominik Peters. Precise complexity of the core in dichotomous and additive hedonic games. In Jörg Rothe, editor, *Algorithmic Decision Theory - 5th International Conference, ADT 2017, Luxembourg, Luxembourg, October 25-27, 2017, Proceedings*, volume 10576 of *Lecture Notes in Computer Science*, pages 214–227. Springer, 2017. doi:10.1007/978-3-319-67504-6_15.

40 Debraj Ray. *A game-theoretic perspective on coalition formation.* Oxford University Press, 2007.

41 Walid Saad, Zhu Han, Tamer Basar, Mérouane Debbah, and Are Hjørungnes. Hedonic coalition formation for distributed task allocation among wireless agents. *IEEE Trans. Mob. Comput.*, 10(9):1327–1344, 2011. `doi:10.1109/TMC.2010.242`.

42 Jakub Sliwinski and Yair Zick. Learning hedonic games. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2730–2736. ijcai.org, 2017. `doi:10.24963/ijcai.2017/380`.

43 Shao Chin Sung and Dinko Dimitrov. On core membership testing for hedonic coalition formation games. *Oper. Res. Lett.*, 35(2):155–158, 2007. `doi:10.1016/j.orl.2006.03.011`.

44 Shao Chin Sung and Dinko Dimitrov. Computational complexity in additive hedonic games. *Eur. J. Oper. Res.*, 203(3):635–639, 2010. `doi:10.1016/J.EJOR.2009.09.004`.

45 Gerhard J. Woeginger. A hardness result for core stability in additive hedonic games. *Math. Soc. Sci.*, 65(2):101–104, 2013. `doi:10.1016/j.mathsocsci.2012.10.001`.

Crossing Number Is NP-Hard for Constant Path-Width (And Tree-Width)

Petr Hliněný
Masaryk University, Brno, Czech Republic

Liana Khazaliya
Technische Universität Wien, Austria

Abstract

Crossing Number is a celebrated problem in graph drawing. It is known to be NP-complete since the 1980s, and fairly involved techniques were already required to show its fixed-parameter tractability when parameterized by the vertex cover number. In this paper we prove that computing exactly the crossing number is NP-hard even for graphs of path-width 12 (and as a result, for simple graphs of path-width 13 and tree-width 9).

Thus, while tree-width and path-width have been very successful tools in many graph algorithm scenarios, our result shows that general crossing number computations unlikely (under P $\neq$ NP) could be successfully tackled using graph decompositions of bounded width, what has been a "*tantalizing open problem*" [S. Cabello, Hardness of Approximation for Crossing Number, 2013] till now.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Computational geometry

Keywords and phrases Graph Drawing, Crossing Number, Tree-width, Path-width

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.40

Related Version *Full Version*: `https://arxiv.org/abs/2406.18933`

Funding Liana Khazaliya acknowledges support from the Austrian Science Fund (FWF) [Y1329] and the European Union's Horizon 2020 COFUND programme [LogiCS@TUWien, grant agreement No.101034440].

1 Introduction

The notion of a crossing number originally arose during WWII by Turán [16] for completed bipartite graphs in the context of the minimization of the number of crossings between tracks connecting brick kilns to storage sites. Formally, the *crossing number* $\mathrm{cr}(G)$ of a graph G is the minimum number of pairwise edge crossings in a drawing of G in the plane. Determining the crossing number of a graph is one of the most prominent combinatorial optimization problems in graph theory.

Back in the 1980s, Garey and Johnson [8] showed that the crossing number minimization is NP-hard. Their result has been extended even to fairly restrictive graph classes, in particular the problem is NP-hard even for cubic graphs [10], and also for a fixed rotation scheme [14]. Moreover, Crossing Number is APX-hard [2] (does not admit a PTAS unless P = NP) in its general setting.

Another direction of the extensive research is on computation of the crossing number for graphs that are initially close to planar graphs. Surprisingly, Crossing Number remains NP-hard for almost planar graphs (graphs that can be made planar and hence crossing-free by the removal of just a single edge) [3], and remains NP-hard on almost planar graphs even when only 3 vertices are of degree greater than 3 [11, 12]. This means that with respect to the maximum degree of the graph, as well as with respect to the number of edges to remove from the graph to make it planar, Crossing Number is para-NP-hard.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 40; pp. 40:1–40:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One more way to deal with the hardness of Crossing Number, is exploiting the structure of the input to get an understanding of how it affects the computational feasibility of the problem. From this side, if the input graph has a vertex cover of bounded size, then the crossing number can be computed exactly in FPT-time (some function of the parameter, i.e. vertex cover number, multiplied by a polynomial of the input size) [13]. Thus, investigation of the Crossing Number for other structural parameters (in particular, feedback vertex set number, tree-depth, path-width, and tree-width) not once was mentioned as an interesting research venue to explore [1, 2, 17]. In this direction, it is known that the problem is solvable in linear time on maximal graphs of path-width 3, admits a 2-approximation algorithm on (general) graphs of path-width 3, and admits a $4w^3$-approximation on maximal graphs of path-width w [1]. For a more involved overview of the results on Crossing Number, we refer the reader to a recent survey by Zehavi [17].

In this paper, we present a hardness result: Crossing Number is NP-hard even on a graphs of constant path-width (and, respectively, tree-width), namely, for path-width 12 (and tree-width 9). That also immediately closes the question of whether Crossing Number is FPT or XP on aforementioned graph classes under usual complexity assumptions, since our result shows that the problem is para-NP-hard.

▶ **Theorem 1.1** (cf. Theorem 3.1 and Theorem 3.2). *Given a graph G and an integer k, the problem to decide whether a graph G can be drawn with at most k crossings is* NP*-complete even when G is required to have path-width at most* 12*, and when G is required to be simple of path-width at most* 13 *and tree-width at most* 9*.*

In Section 2 we define the basic concepts, i.e., of a drawing, the crossing number, width decompositions, and the problem itself. In Section 3 we describe a hardness reduction from Satisfiability. Since the proof is rather technical, we propose separately the construction (Section 3.3), necessary conditions for an existence of a drawings with some predefined crossing number (Section 3.4), correctness of the reduction (Section 3.5), and, lastly, that the width parameters, i.e. path-width and tree-width, of the constructed graph are constant (Section 3.6). We conclude with Section 4.

2 Preliminaries

We will consider finite graphs with possible parallel edges throughout the paper. We begin with the standard terminology of graph theory [7], including the notions of tree-width and path-width [5] which are commonly used parameters to capture the complexity of a graph, and of graph drawing concepts [6].

Furthermore, for an integer $n \in \mathbb{N}$, we denote by $[n] = \{1, \ldots, n\}$.

2.1 Drawings

A *drawing* $\mathcal{G}$ of a graph G in the plane is a mapping of the vertices $V(G)$ to distinct points in the plane, and of the edges $E(G)$ to simple curves connecting their respective endpoints but not containing any other vertex point. When convenient, we will refer to the elements (vertices and edges) of the drawing by the corresponding elements of G. A *crossing* is the intersection (a common point) of two distinct edge curves, other than their common endpoint. It is well established that the search for an optimal solution to the crossing number problem can be restricted to so called *good drawings*: any pair of edges crosses at most once, adjacent edges do not cross, and there is no crossing point in common to three or more edges.

A drawing $\mathcal{G}$ is *planar* (or a *plane graph*) if $\mathcal{G}$ has no crossings, and a graph in *planar* if it has a planar drawing. The number of crossings in a particular drawing $\mathcal{G}$ is denoted by $\mathrm{cr}(\mathcal{G})$ and the minimum over all good drawings $\mathcal{G}$ of a graph G by $\mathrm{cr}(G)$. We call $\mathrm{cr}(\mathcal{G}_G)$ and $\mathrm{cr}(G)$ the *crossing number* of the drawing $\mathcal{G}_G$ and the graph G, respectively. The Crossing Number problem for a given graph G asks for a good drawing $\mathcal{G}$ with the least possible number of crossings.

We will also use a common artifice in crossing number research. In a *weighted* graph, each edge is assigned a positive number (the *weight* or thickness of the edge, usually an integer). Now the *weighted crossing number* is defined as the ordinary crossing number, but a crossing between edges e_1 and e_2, say of weights t_1 and t_2, contributes the product $t_1 \cdot t_2$ to the weighted crossing number. For the purpose of computing the crossing number, an edge of integer weight t can be equivalently replaced by t parallel edges of weights 1; this is since we can easily redraw each of the t edges closely along one with the least number of crossings. Hence, from now on, we will use weighted edges instead of parallel edges, and shortly say *crossing number* to the weighted crossing number.

2.2 Tree-width and Path-width

A *tree decomposition* $\mathcal{T}$ of an undirected graph G is a pair (T, χ), where T is a tree (whose vertices we call *nodes*) rooted at a node r and χ is a function that assigns to each node $t \in \mathcal{T}$ a set $\chi(t) \subseteq V(G)$ such that the following holds:

- For every $\{u, v\} \in E(G)$ there is a node t such that $u, v \in \chi(t)$.
- For every vertex $v \in V(G)$, the set of nodes t satisfying $v \in \chi(t)$ forms a nonempty subtree of T.

The sets $\chi(t)$, for $t \in V(T)$, are called *bags* of the tree decomposition. The *width* of a tree decomposition (T, χ) is the size of a largest set $\chi(t)$ minus 1, and the *tree-width* of the graph G, denoted $\mathrm{tw}(G)$, is the minimum width of a tree decomposition of G.

The *path decomposition* and *path-width* are defined analogously with the only difference that the tree T is required to be a path.

We are going to use the following cops-and-robber game characterization on the graph G.

- The *robber* player can freely move along cop-free paths in the graph.
- The *cops* fly in a helicopter; can land on a vertex or be lifted back up. When the helicopter shows above a vertex v, the robber has time to escape wherever they chooses to.
- The robber is caught whenever a cop lands on the robber's vertex v.

Such a game is called *monotone* if the robber never gets a chance to reach a vertex previously occupied by a cop.

The cited characterization is as follows.

▶ **Theorem 2.1** (Seymour and Thomas [15]).

(1) *The tree-width of G is at most t if and only if $t+1$ cops can always catch the robber in G in a monotone game if the robber is visible to the cop player.*

(2) *The path-width of G is at most t if and only if $t+1$ cops can always catch the robber in G in a monotone game provided the robber is* not *visible to the cops.*

3 Hardness Reduction

In this section, we present and prove a polynomial time reduction that given an instance $\mathcal{I} = (\mathcal{C}, \mathcal{V})$ of Satisfiability, constructs an equivalent instance (G, k) of Crossing Number on a graph of constant path-width (and tree-width).

Satisfiability
Input: A set of clauses $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ over variables $\mathcal{V} = \{x_1, \ldots, x_n\}$
Question: Does there exist an assignment of the variables $\tau : \mathcal{V} \to \{\texttt{True}, \texttt{False}\}$ satisfying all clauses in $\mathcal{C}$?

Crossing Number
Input: A graph G, and $k \in \mathbb{Z}_{\geq 0}$
Question: Does G admit a drawing $\mathcal{G}$ in the plane such that $\mathcal{G}$ has at most k crossings?

▶ **Theorem 3.1.** *There is a polynomial-time algorithm that, given an instance $\mathcal{I}$ of Satisfiability, outputs an equivalent instance of Crossing Number on a graph G of path-width at most* 12 *and tree-width at most* 9 *(where G is allowed to have parallel edges).*

If simplicity of the graph G is desirable, we immediately conclude:

▶ **Corollary 3.2.** *There is a polynomial-time algorithm that, given an instance $\mathcal{I}$ of Satisfiability, outputs an equivalent instance of Crossing Number on a simple graph G of path-width at most* 13 *and tree-width at most* 9*.*

Proof. For any graph G and $e \in E(G)$, the same drawing as a point set may be used both for G and for G with the edge e subdivided; informally, subdivisions of edges do not matter for Crossing Number. Hence, if the graph G of Theorem 3.1 contains parallel edges, we form a graph G' by subdividing each such edge of G once and obtain $\mathrm{cr}(G') = \mathrm{cr}(G)$. The tree-width does not change, and the path-width of G' grows by at most 1 compared to G. ◀

3.1 High-level Idea

Naturally, for interpreting a Satisfiability instance $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ in an instance (G, k) of Crossing Number, one would use a large "grid structure". Such structure would allow to separately interpret values of the variables $\mathcal{V}$, and to let all clauses $\mathcal{C}$ interact with their variables; one could imagine, e.g., variables in columns and clauses in rows of the grid structure, and their interaction happening in specially crafted cells in which the row and the column intersect.

However, if a graph contains a large grid as a minor, then its tree-width is also large, and our aim here is to obtain a graph G of constant tree-width and path-width. Thus, we are instead going to base our reduction on a frame graph F with many small separators (here of size $4 + 4$) ordered from left to right, in order to achieve constant path-width of resulting G. The crucial thing is that for each of the separators X, there are three components of $(F - X)$ – the "left", "middle" and "right" ones – such that the left and right components are *forced to cross* with the middle component many times (see Figure 1 and Figure 2 for a brief illustration). This way we enforce the sought large grid structure in any optimal drawing of the frame F, and consequently in any optimal drawing of G.

At the same time, the frame is constructed such that there is certain drawing flexibility possible, namely we can perform "vertical flips" of the middle components of separators mentioned in the previous paragraph (see Figure 1), and these will form a part of the variable gadgets in our reduction. We will use this drawing flexibility of our variable gadgets to encode the truth values of variables in Satisfiability (see Figure 4 for a brief illustration of this encoding). Specific small gadgets (see Figure 3) will be added to the variable gadgets in G to encode in which clauses they participate, and a satisfying assignment of the variables will then be checked as a possibility to draw added global clause edges of G (one edge per

Table 1 Color-encoding of the weights of the corresponding edges; and $\Theta_{n,\ell}(\omega^1)$ denotes the class of functions f such that $C_1 \cdot \omega^1 \leq f(\omega^1) \leq C_2 \cdot \omega^1$ for positive constants C_1, C_2 dependent on n and ℓ, but not on ω.

Color	Usage	Weight
Heavy-brown (HB)	The frame and Var-gadgets attachments	ω^8
Light-black (LB)	Var-gadgets interior skeletons	ω^6
Red (R)	Paths in Var-gadgets (vertical)	$\omega^4 + \Theta_{n,\ell}(\omega^1)$
(R')	Stairs interconnecting Var-gadgets (horizontal)	ω^3
Blue (B)	Paths in Var-gadgets (vertical)	$\omega^4 + \Theta_{n,\ell}(\omega^1)$
(B')	Stairs connecting within Var-gadgets (horizontal)	ω^3
Cyan (C)	Clauses Encoding within Var-gadgets	ω^2
Green (G)	(Global) Clause Edges	$\omega^0 = 1$

each clause, see the green edges in Figure 2) with minimum crossing cost across the whole picture. This is an idea similar to the one used in [3]. The crucial point of the construction, however, is how to enforce the unique right crossing pattern between the frame components as in Figure 1, and for this we build upon an idea originally introduced in [12] and now detailed within Section 3.4.

3.2 Auxiliary Graphs

To facilitate the presentation, we use colors, i.e. heavy-brown (HB), light-black (LB), red (R), blue (B), cyan (C), green (G), to encode the future order of the weights of the corresponding edges (see in Section 3.2). The weights of the edges will play a crucial role in the future description of possible drawings of the constructing graph. The weight values are assigned with respect to a sufficiently large (still polynomial in $|\mathcal{C}| + |\mathcal{V}|$) edge weight ω, e.g., $\omega = |E(G)|^2$. Then, informally, even one crossing of weight ω^{t+1} "outweighs" all crossings of G of weight ω^t. Observe that, importantly, all weights used in our construction will be bounded by a polynomial in $|\mathcal{I}|$.

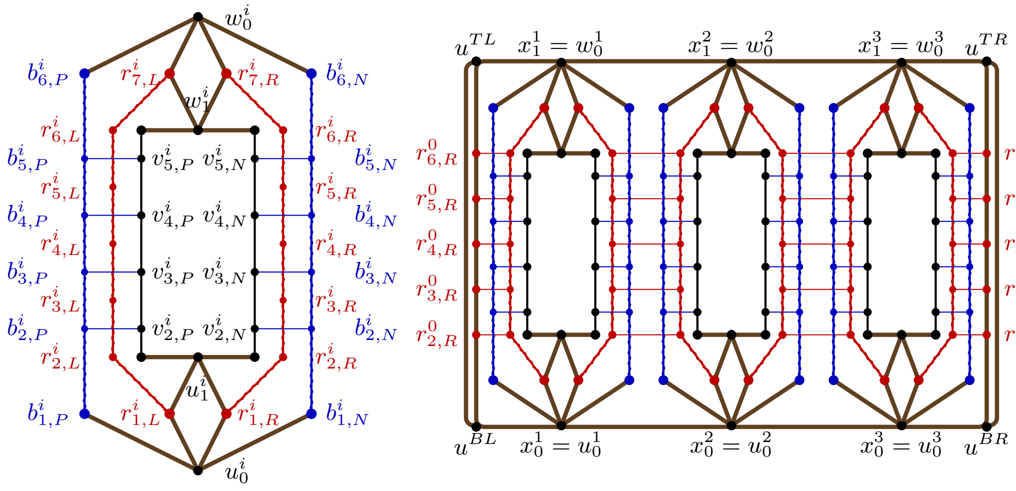
Further, we present auxiliary graphs for use as building blocks (Figure 1), before describing the whole construction of the crossing number instance G.

Variable gadgets. We start by defining Var-*gadgets*. For each $i \in [n]$, we construct a Var^i-gadget of height $h \in \mathbb{Z}_{>0}$ (see an example of Var^i for $h = 4$ in Figure 1a, the value of h to be defined later).

- First, we introduce the vertex set of Var^i as

$$V(\mathsf{Var}^i) = \{b^i_{j,P}, b^i_{j,N}, v^i_{j,P}, v^i_{j,N}\}_{j\in[h+2]} \cup \{r^i_{j,L}, r^i_{j,R}\}_{j\in[h+3]} \cup \{w^i_0, u^i_0, w^i_1, u^i_1\}.$$

- We add 6 paths as follows:
  - two B-paths (constructed on B-edges) go through vertices $\{b^i_{j,P}\}_{j\in[h+2]}$ and $\{b^i_{j,N}\}_{j\in[h+2]}$, and we will refer to the paths as B-pos and B-neg respectively;
  - two LB-paths go through vertices $\{v^i_{j,P}\}_{j\in[h+2]}$ (LB-pos) and $\{v^i_{j,N}\}_{j\in[h+2]}$ (LB-neg);
  - two R-paths go through vertices $\{r^i_{j,L}\}_{j\in[h+3]}$ (R-left) and $\{r^i_{j,R}\}_{j\in[h+3]}$ (R-right).
- We make these paths adjacent (with HB-edges) to the vertices $w^i_0, u^i_0, w^i_1, u^i_1$ as follows:
  - both B-pos and B-neg paths by their corner vertices to w^i_0 and u^i_0;



(a) The variable gadget Var^i, $h = 4$.

(b) The frame with n variable gadgets for $n = 3$, $h = 4$.

Figure 1 Auxiliary graphs. Note that for each $i \in [n]$, the 8-tuple $\{u_0^i, u_1^i, r^i_{1,L}, r^i_{1,R}, w_0^i, w_1^i, r^i_{h+3,L}, r^i_{h+3,R}\}$ is a vertex cut in the frame graph.

  - both `LB-pos` and `LB-neg`, analogously, to w_1^i and u_1^i; and
  - both `R-left` and `R-right` paths to both w_0^i, u_0^i and w_1^i, u_1^i (see Figure 1a).
- Lastly, for each $j \in [2, h+1]$, we add *stairs* between pairs of `-pos` and `-neg` paths, i.e., pairwise connecting `B-`/`LB-pos` and `B-`/`LB-neg` paths with `B'`-edges $b^i_{j,P}v^i_{j,P}$ and $b^i_{j,N}v^i_{j,N}$ respectively.
- The *weight* of each edge of Var^i is as specified in Section 3.2; in particular, for the `R`-paths, the weight of the edges $r^i_{j,L}r^i_{j+1,L}$ and $r^i_{j,R}r^i_{j+1,R}$ is exactly $\omega^4 + j(j+1)\omega$, and for the `B`-paths, the weight of the edges $b^i_{j,P}b^i_{j+1,P}$ and $b^i_{j,N}b^i_{j+1,N}$ is exactly $\omega^4 + j(j+2)\omega$.

The frame. We construct the *frame* for n Var-gadgets of height h, where $n, h \in \mathbb{Z}_{>0}$.

- First, we introduce the `HB`-cycle (with `HB`-edges) on 4 vertices u^{BL} (bottom-left), u^{TL} (top-left), u^{TR} (top-right), u^{BR} (bottom-right) in the specified order.
- Then, we subdivide (n times) the edge between u^{BL} and u^{BR} by adding vertices $\{x_0^i\}_{i\in[n]}$; analogously we subdivide the edge between u^{TL} and u^{TR} by adding $\{x_1^i\}_{i\in[n]}$.
- Further, we add another `HB`-edge between u^{BL} and u^{TL} (resp., between u^{TR} and u^{BR}) and subdivide it h times by adding vertices $\{r^0_{j,R}\}_{j\in[2,h+2]}$ (resp., by adding $\{r^{n+1}_{j,L}\}_{j\in[2,h+2]}$).

We call the resulting graph of this construction (see Figure 1b) the *frame* F.

Now, we attach n Var-gadgets to the frame F.

- For each $i \in [n]$, we introduce a Var^i-gadget (as described in Section 3.2) and pairwise identify vertices u_0^i, w_0^i of Var^i with vertices x_0^i, x_1^i of the frame respectively.
- Lastly, we add *stairs* (interconnections) between `R`-paths of the neighboring Var-gadgets and the frame, i.e. for each $i \in [n+1]$ and $j \in [2, h+2]$, we add `R'`-edge $r^{i-1}_{j,R}r^i_{j,L}$. Thus, for each $i \in [n+1]$, we make stairs between the `R-right` path of Var^{i-1} (or, if $i = n+1$, with a subdivision of the frame's side $u^{TR}u^{BR}$) and the `R-left` path of Var^i (or, if $i = 1$, with a subdivision of the frame's side $u^{TL}u^{BL}$).

The weights of all new edges are again as specified in Section 3.2.

This finishes the construction of our *frame with n* Var-*gadgets* G' (see G' for $h = 4$, $n = 3$ in Figure 1b). Note that G' still lacks the clause edges (see in further Figure 2) and an interpretation of variable occurrences in clauses (the cells of further Figure 3).

So far, for simplicity, we allow ourselves to refer to Figure 1b to illustrate the defined graph G'. Observe the natural meaning the `R-left` and `R-right` paths in each gadget Var^i; in order to facilitate their connections to Var^{i-1} and to Var^{i+1}, `R-left` is naturally drawn to the left of `R-right`. On the other hand, the `B-/LB-pos` and `B-/LB-neg` paths of Var^i are symmetric and not adjacent outside of Var^i, and hence they can be flexibly drawn – `B-/LB-pos` to the left or to the right of `B-/LB-neg`; this is what will later define the truth value of the variable represented by Var^i.

3.3 The Full Reduction

Consider an instance $(\mathcal{C}, \mathcal{V})$ of Satisfiability (with $|\mathcal{C}| = \ell$, $|\mathcal{V}| = n$). We construct an instance (G, k) of Crossing Number as follows. See Figure 2 for a schematic representation.

- First, we introduce G', the frame with n Var-gadgets of height $h = 4\ell + n - 2$.
- Then, for each $i \in [n]$, we encode the occurrence of the variable x_i in clauses $\mathcal{C}$. For that purpose, for each $j \in [\ell]$, between `LB-pos` and `LB-neg` paths of the existing Var^i-gadget we add a *cell*. Each cell is defined by two horizontal `LB`-edges and 3 edges inside, depending on the type (pos, neg, neut) of the cell: pos if $x \in C$; neg if $\overline{x} \in C$; neut if neither x nor $\overline{x}$ is in C. A cell of each type is shown in Figure 3.

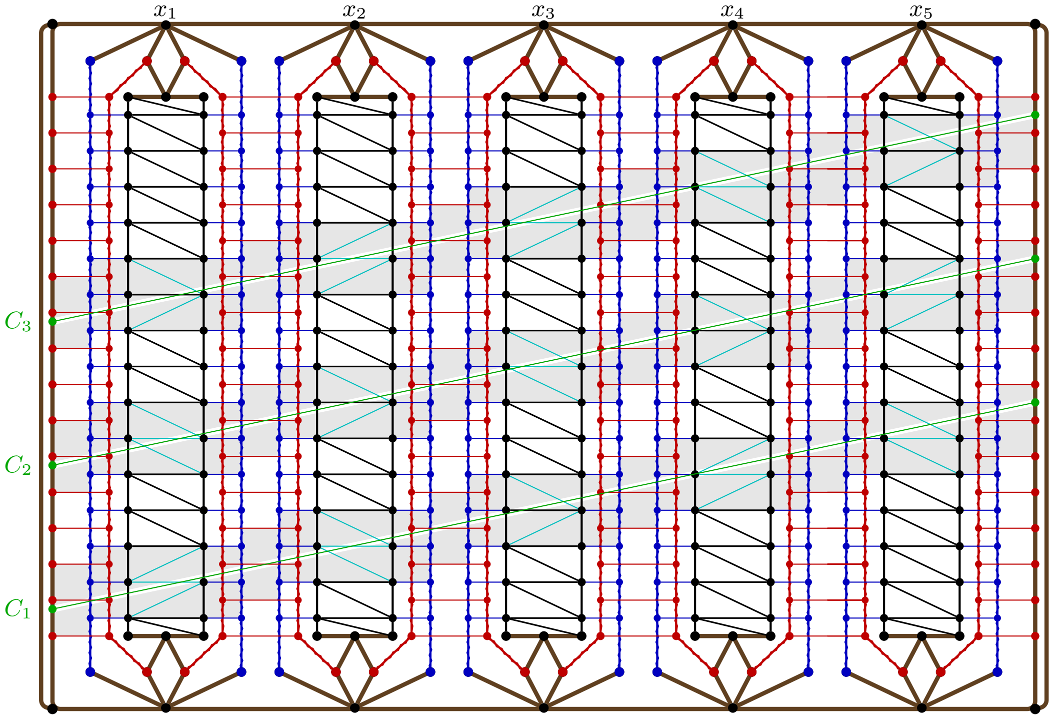
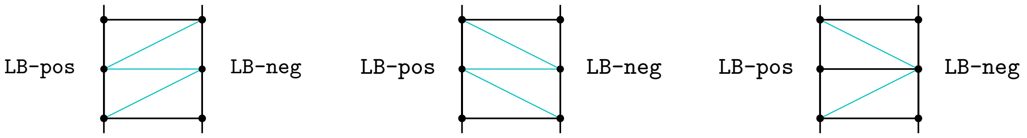


Figure 2 For an example instance of Satisfiability, given by $\mathcal{V} = \{x_1, x_2, x_3, x_4, x_5\}$ and $\mathcal{C} = \{(x_1 \vee \overline{x_2} \vee x_4 \vee \overline{x_5}), (\overline{x_1} \vee \overline{x_3} \vee x_5), (x_2 \vee x_3 \vee \overline{x_4})\}$; the depicted graph G is constructed as an input of the sought reduction to Crossing Number of G. Notice, in particular, the addition of the clause edges (drawn in green from left to right across the frame) and the shaded areas in which the clause edges will presumably be drawn.



(a) C_{pos}: $x \in C$. **(b)** C_{neg}: $\overline{x} \in C$. **(c)** C_{neut}: neither x nor $\overline{x}$ is in C.

Figure 3 Cell types; for cases of variable x occurrence in clause C.

Cells inside the same Var^i-gadget are separated with additional LB-edges as shown in Figure 2. For a formal description, we start with LB-edges (again, all weights are as specified in Section 3.2) inside Var-gadgets which separate cells. For each $i \in [n]$, we add

- an LB-path from $v^i_{1,N}$ to $v^i_{1+i,P}$ (below all cells of this Var^i-gadget), precisely on vertices $v^i_{1,N}$, $v^i_{2,P}$, $v^i_{2,N}$, $\dots$, $v^i_{i,N}$, $v^i_{1+i,P}$ in this order;
- another LB-path from $v^i_{4\ell+i-1,N}$ to $v^i_{h+2,P}$ (above all cells in Var^i), precisely on vertices $v^i_{4\ell+i-1,N}$, $v^i_{4\ell+i,P}$, $v^i_{4\ell+i,N}$, $\dots$, $v^i_{h+1,N}$, $v^i_{h+2,P}$; and
- for each $j \in [\ell - 1]$, a LB-path on vertices $v^i_{4j+i-1,N}$, $v^i_{4j+i,P}$, $v^i_{4j+i,N}$, $v^i_{4j+i+1,P}$ (between cells number j and $j+1$).

After that, we add cells themselves in the bottom-up order. For all $j \in [\ell]$, we introduce two LB-edges $v^i_{4j+i-3,P}v^i_{4j+i-3,N}$ and $v^i_{4j+i-1,P}v^i_{4j+i-1,N}$, and then we proceed with encoding of our Satisfiability instance $(\mathcal{C}, \mathcal{V})$ it the following way:

- if $x_i \in C_j$, we introduce a C_{pos} cell (Figure 3a), i.e. we add three C-edges as a path through $v^i_{4j+i-3,P}$, $v^i_{4j+i-2,N}$, $v^i_{4j+i-2,P}$, $v^i_{4j+i-1,N}$;
- in case $\overline{x_i} \in C_j$, we introduce a C_{neg} cell (Figure 3b), i.e. we add three C-edges as a path through $v^i_{4j+i-3,N}$, $v^i_{4j+i-2,P}$, $v^i_{4j+i-2,N}$, $v^i_{4j+i-1,P}$;
- lastly, if neither x_i nor $\overline{x_i}$ is in C_j, we introduce a C_{neut} cell (Figure 3c), formed by one LB-edge $v^i_{4j+i-2,P}v^i_{4j+i-2,N}$ and two C-edges $v^i_{4j+i-3,P}v^i_{4j+i-2,N}$ and $v^i_{4j+i-2,N}v^i_{4j+i-1,P}$.

- Finally, for each $j \in [\ell]$, we add a G-edge that corresponds to the clause C_j itself. Here, we subdivide two vertical HB-edges of the frame and connect these newly added vertices. Precisely, for each $j \in [\ell]$, we subdivide the edge between $r^0_{4j-2,R}$, $r^0_{4j-1,R}$ (on the left vertical side $u^{BL}u^{TL}$ of the frame, cf. Figure 1b) and the edge between $r^{n+1}_{4j+n-1,L}$, $r^{n+1}_{4j+n,L}$ (on the right vertical side $u^{BR}u^{TR}$). Note the shift in the indices of the subdivided edges, up by $n+1$ from left to right.

This concludes the construction of G.

The reduction returns (G, k) as an instance of Crossing Number where, for $h = 4\ell + n - 2$,

$$k = 2n(2h+1)\omega^7 + 2n\ell\omega^6 + 4n\ell\omega^4 + 2n\sum_{j=2}^{h+1} j(j+1)\omega^4 + 2n\sum_{j=1}^{h+1} j(j+2)\omega^4 + n\ell\omega^2 + (\omega^2 - 1).$$

3.4 Drawings Claims

Until this point, all Figures were provided as an illustration without arguing why a certain drawing of the corresponding graph was selected. This subsection is dedicated to shed light on the conditions that necessarily have to be satisfied for a drawing $\mathcal{G}$ of the previously constructed instance (G, k) of the Crossing Number unless $\mathcal{G}$ is not a solution.

So, we are considering the constructed instance (G, k) of Crossing Number from Section 3.3. Following the way the graph G was introduced, we begin to formulate observations and claims that every drawing of the graph with at most k crossings has to satisfy. Naturally,

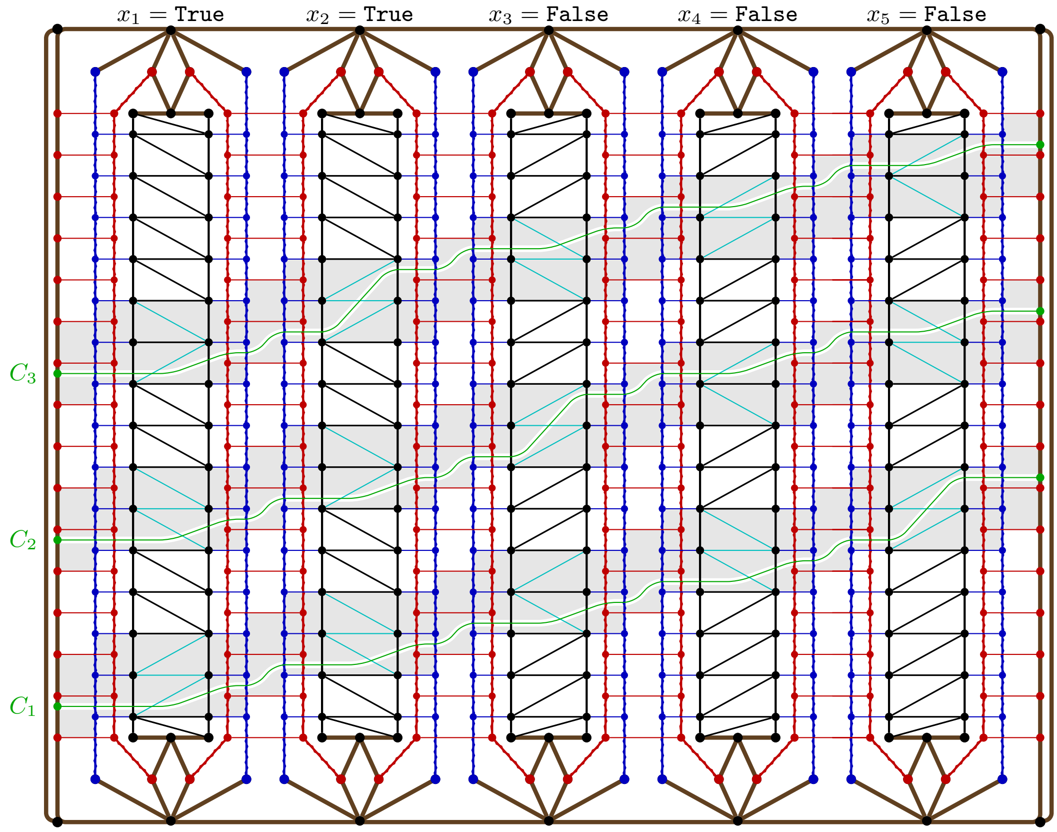


Figure 4 A drawing of the graph G from Figure 2, constructed from an instance of SATISFIABILITY given by $\mathcal{V} = \{x_1, x_2, x_3, x_4, x_5\}$ and $\mathcal{C} = \{(x_1 \vee \overline{x_2} \vee x_4 \vee \overline{x_5}), (\overline{x_1} \vee \overline{x_3} \vee x_5), (x_2 \vee x_3 \vee \overline{x_4})\}$. The depicted drawing $\mathcal{G}$ of G corresponds to the satisfying assignment $x_1 = x_2 = \texttt{True}$, $x_3 = x_4 = x_5 = \texttt{False}$. The clause $C_1 = (x_1 \vee \overline{x_2} \vee x_4 \vee \overline{x_5})$ is satisfied by the variable x_5 (observe that the G-edge of C_1 makes an extra jump-up in the drawing area of Var^5 to the right, yet crossing only one C-edge there – same as in other gadgets), $C_2 = (\overline{x_1} \vee \overline{x_3} \vee x_5)$ is satisfied by x_3 and $C_3 = (x_2 \vee x_3 \vee \overline{x_4})$ is satisfied by x_2. See Figure 5 for a drawing representing an unsatisfying assignment.

we start with conditions for the heaviest edges, for the frame and Var-gadgets, and after that we argue about the clauses encoding. Due to space restrictions, we leave proofs of the (*)-marked statements to the full arXiv version of the paper.

▶ **Observation 3.3.** **(*)** *If there exists a drawing $\mathcal{G}$ of G such that $\mathrm{cr}(\mathcal{G}) \leq k$, then $\mathcal{G}$ has no crossing that involves any of HB-edges.*

▶ **Observation 3.4.** **(*)** *Let H is a subgraph of G. If (H, k) is a no-instance of the CROSSING NUMBER, then (G, k) is a no-instance of the CROSSING NUMBER.*

Based on Theorem 3.4, we now show some claims that speak only about a subgraph of G. As in Section 3.2, let F be the frame and G' be the frame with n Var-gadgets (Figure 1) of the graph G; so, we are going to speak about properties which hold regardless of our clauses encoding in G. However, both next claims still follow from the same type of argument, namely, any crossing between the considered edges would be more costly than the selected value of k allows.

▶ **Claim 3.5.** **(*)** *If there exists a drawing $\mathcal{G}'$ of G' such that $\mathrm{cr}(\mathcal{G}') \leq k$, then there are no other crossings than crossings between R- or B-edges with R'- or B'-edges in $\mathcal{G}'$.*

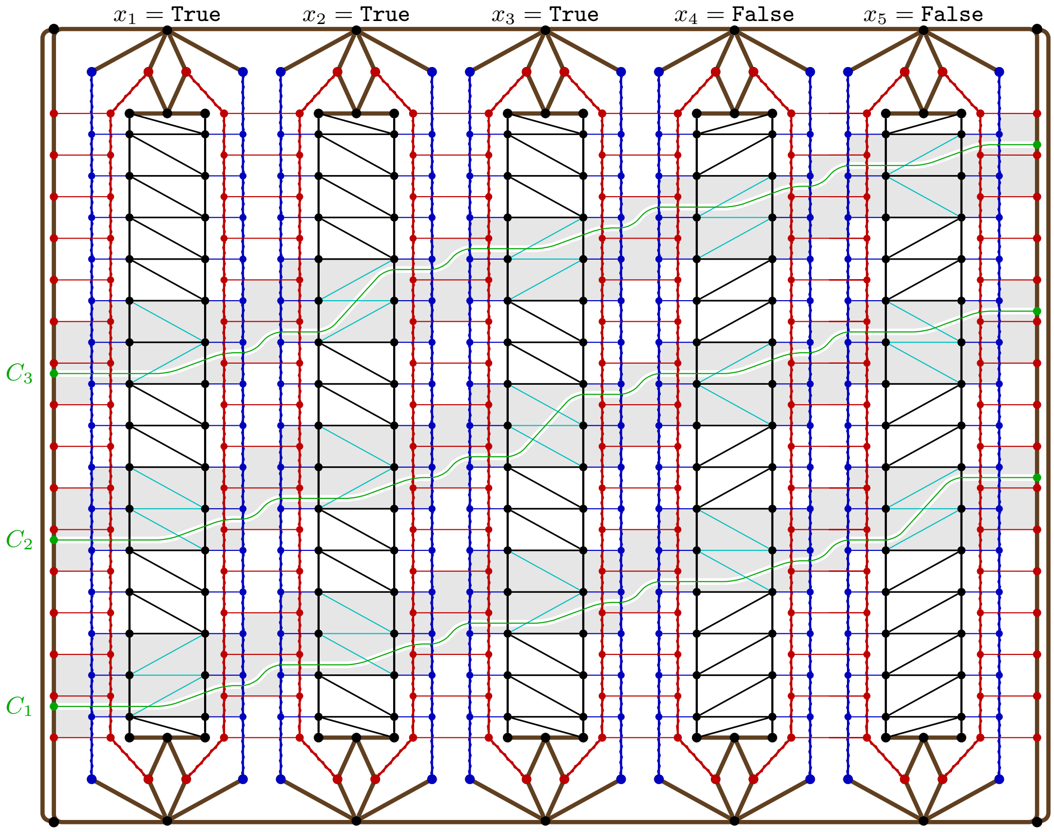


Figure 5 Another drawing of the graph G from Figure 2, constructed from an instance of Satisfiability given by $\mathcal{V} = \{x_1, x_2, x_3, x_4, x_5\}$ and $\mathcal{C} = \{(x_1 \vee \overline{x_2} \vee x_4 \vee \overline{x_5}), (\overline{x_1} \vee \overline{x_3} \vee x_5), (x_2 \vee x_3 \vee \overline{x_4})\}$, for comparison with the drawing in Figure 4.
The depicted drawing $\mathcal{G}'$ of G corresponds to the unsatisfying assignment $x_1 = x_2 = x_3 =$ True, $x_4 = x_5 =$ False. The clause $C_1 = (x_1 \vee \overline{x_2} \vee x_4 \vee \overline{x_5})$ is satisfied by the variable x_5 and $C_3 = (x_2 \vee x_3 \vee \overline{x_4})$ is satisfied by x_2, same as in Figure 4. The difference for the unsatisfied clause $C_2 = (\overline{x_1} \vee \overline{x_3} \vee x_5)$ is that here the G-edge of C_2 has no way to make the required extra jump-up without crossing more than one C-edge inside one of Var-gadgets (or other heavier edges). Hence, here the G-edge of C_2 makes extra crossing with two C-edges in the drawing area of Var^3, and this unavoidably leads to $\mathrm{cr}(\mathcal{G}') > k$.

▶ **Claim 3.6.** **(*)** *If a drawing $\mathcal{G}'$ of G' is a solution of the instance (G', k), then, for all $u, v \in V(G' \setminus F)$, both u and v are lying in the same face of the frame F in $\mathcal{G}'$, and the selection of such a face is uniquely predetermined.*

Theorem 3.6 allows us, without loss of generality, fix a drawing of the frame F as it is shown in Figure 1b. So, we define a positioning of left and right sides. This way, the fixed drawing of the frame determines a linear order of gadgets Var^i (from left to right following increasing order of $i \in [n]$).

Furthermore, we already almost fixed a drawing of Var-gadgets inside the frame. Still, the fact that we have not enough budget for any other crossing (namely, crossings between R'- and B'-edges of weight ω^3 have not been covered yet) needs some proper counting that we will provide further. By now, according to Theorem 3.5, the R/B/LB-paths of Var-gadgets are not allowed to cross each other in any solution if such a one exists.

Lastly, let us notice that the positioning of the vertical R-paths is also fixed.

▶ **Claim 3.7.** **(*)** *If a drawing $\mathcal{G}'$ of G' is a solution of the instance (G', k), then, for all $i \in [n]$, the `R-left` path of the* Var^i *is drawn to the left from both* `LB`*-paths of* Var^i*, while the* `R-right` *path is to the right.*

Now, let us check how crossings of `R`- and `B`-edges with `R'`- and `B'`-edges behave. Briefly saying, by the following Theorem 3.8, we show that crossings between the `R`- and `B`-edges are also predefined by a construction. For this purpose, while constructing the graph G, we played a bit with additional *adjustment* weight of order ω^1 for the vertical `R`- and `B`-edges. This adjustment weight selection forces the unique alternation of crossings exactly as shown in Figure 1b and subsequent figures.

▶ **Lemma 3.8.** **(*)** *If a drawing $\mathcal{G}'$ of G' is a solution of the instance (G', k), then each* `B`*-edge (*`R`*-edge) crosses exactly one* `R'`*-edge (*`B'`*-edge). The total weight of all crossings between* `B(R)`*- and* `R'(B')`*-edges adds* $2n\left((2h+1)\omega^7 + \sum_{j=2}^{h+1} j(j+1)\omega^4 + \sum_{j=1}^{h+1} j(j+2)\omega^4\right)$ *to the count* $\mathrm{cr}(\mathcal{G}')$*.*

Next, we return back to the full reduction graph G, and investigate properties of its admissible drawings.

▶ **Lemma 3.9.** **(*)** *If the crossing number of a drawing $\mathcal{G}$ of G does not exceed k, then the* `G`*-edges add at least $2n\ell\omega^6 + 4n\ell\omega^4 + n\ell\omega^2$ more crossings. In particular, every* `G`*-edge in $\mathcal{G}$ crosses precisely one* `C`*-edge of every* Var^i*-gadget.*

For each `G`-edge, we define its *cells area* as a connected region (union) of faces in a drawing $\mathcal{G}$ of G as it is shown in Figure 4 with a grey fill, i.e. for each $j \in [\ell]$, there is a single connected horizontal block of faces that includes all cells of the clause C_j and will correspond to the cells area for the `G`-edge which is j^{th} in bottom-up order. For each such a cells area we furthermore define its *up*- and *down-level* as the subsets of faces that are higher and lower, respectively, by subscripts of their vertices. To clarify the last point, by the next lemma we show that for each $j \in [\ell]$, if the crossing number of the drawing $\mathcal{G}$ of the graph G does not exceed k, then the j^{th} `G`-edge does not leave its cells area. Now, we proceed with that formally.

▶ **Lemma 3.10.** **(*)** *If the crossing number of a drawing $\mathcal{G}$ of G does not exceed k, then any* `G`*-edge cannot be drawn (even partially) outside of its cells area (see Figure 4).*

3.5 Correctness

Before proceeding with the correctness arguments, let us look back and see that, indeed, all drawing claims imply that if some drawing $\mathcal{G}$ of G with $\mathrm{cr}(\mathcal{G}) \leq k$ exists, then almost all possible crossings are predefined. Practically the only freedom still left is the possibility to *flip* (independently of others) each Var-gadget. By *flips* of the Var-gadget we mean its two possible embeddings, which differ only by the order of `B`- and `LB`-paths, e.g. going from left to right in our Figures. As an example, let us consider Figure 4: here, we meet first from left

- either a pair of `B-pos, LB-pos` paths and then call such an embedding a *pos-side flip* (see Var^3-gadget in the middle in Figure 5);
- or a pair of `B-neg, LB-neg` paths and then call such an embedding a *neg-side flip* (see Var^3-gadget in the middle in Figure 4).

And this is exactly the intuition behind transferring a possible variable assignment from Satisfiability instance to a possible drawing of the Crossing Number instance, and back.

Suppose, given an instance $(\mathcal{C}, \mathcal{V})$ of Satisfiability, that the reduction from the previous subsection returns (G, k) as an instance of Crossing Number.

▶ **Lemma 3.11.** *If $(\mathcal{C}, \mathcal{V})$ is a satisfiable instance of* Satisfiability*, then the graph G of the constructed instance (G, k) admits a drawing $\mathcal{G}$ such that $\mathrm{cr}(\mathcal{G}) \leq k$.*

Proof. Again, let G' be the frame with n Var-gadgets which is a subgraph of G. We start with the drawing of G' as specified by Figure 1, which actually corresponds to the "minimal drawing" investigated in Theorem 3.8. We are going to show that this drawing has exactly the same number of crossings as claimed in Theorem 3.8.

Indeed, for every $i \in [n]$, the `R-left` and `R-right` paths of the Var^i-gadget each are crossed by h `B'`-edges in our drawing, and the `B-pos` and `B-neg` paths of the Var^i-gadget each are crossed by $(h+1)$ `R'`-edges. Taking into account the alternating order of these crossings and the exact weights of the edges of these paths (see in Section 3.2), we count exactly $2\omega^3 \left(h\omega^4 + (h+1)\omega^4 + \sum_{j=2}^{h+1} j(j+1)\omega^1 + \sum_{j=1}^{h+1} j(j+2)\omega^1 \right)$ crossings on Var^i. Summing over all $i \in [n]$, we get $2n \left((2h+1)\omega^7 + \sum_{j=2}^{h+1} j(j+1)\omega^4 + \sum_{j=1}^{h+1} j(j+2)\omega^4 \right)$ crossings. There are no more crossings in our drawing of G'.

Now we add the cell gadgets of G to every Var-gadget, without additional crossings, and the `G`-edges denoted by e_j of all clauses $C_j \in \mathcal{C}$. The goal is to show that G is (can be) drawn with at most k crossings. Each `G`-edge e_j crosses every Var-gadget in total weight of $2\omega^6 + 4\omega^4 + \Theta_{n,\ell}(\omega^1)$ crossings, together giving another $2n\ell\omega^6 + 4n\ell\omega^4 + \Theta_{n,\ell}(\omega^1)$ crossings in our drawing. It only remains to estimate the weight of crossings of the `G`-edges e_j with the cell gadgets. If we can achieve the state that each `G`-edge e_j additionally crosses only one `C`-edge (of weight ω^2) of cell gadgets in every Var-gadget, then, comparing the full count to the reduction parameter definition

$$k = 2n(2h+1)\omega^7 + 2n\ell\omega^6 + 4n\ell\omega^4 + 2n\sum_{j=2}^{h+1} j(j+1)\omega^4 + 2n\sum_{j=1}^{h+1} j(j+2)\omega^4 + n\ell\omega^2 + (\omega^2 - 1),$$

we see that it is enough to have $\Theta_{n,\ell}(\omega^1) \leq \omega^2 - 1$. The latter holds true by our (sufficiently large) choice of the base weight ω.

It thus remains to show that aforementioned desired drawings of the `G`-edges e_j are simultaneously possible. Since the given Satisfiability instance is satisfiable, there is a truth assignment τ of $\mathcal{V}$ such that for every clause $C_j \in \mathcal{C}$, an occurrence of some variable $x_i \in \mathcal{V}$ in C_j satisfies C_j in τ. If the variable $x_i \in \mathcal{V}$, $i \in [n]$, is valued `True`, then the Var^i-gadget is pos-side flipped in our drawing of G', i.e., that the `LB-pos` path of Var^i is to the left of the `LB-neg` path of Var^i. If x_i is `False`, then the Var^i-gadget is neg-side flipped, i.e., with `LB-pos` to the right.

Now, for every $C_j \in \mathcal{C}$, the edge e_j is drawn in the down-level of its cells area (as defined in Section 3.4), until we reach the cell in the Var^i-gadget where i is the least index such that an occurrence of x_i satisfies C_j in τ. Then, inside this Var^i-gadget, the edge e_j jumps up to the up-level of its cells area, as depicted in Figure 3a if $x_i \in C_j$, resp. in Figure 3b if $\overline{x_i} \in C_j$. The Figure shows that this jump-up is also possible with crossing only one `C`-edge in the cells gadget of Var^i. For the rest, the edge e_j is drawn in the up-level of its cells area. This is the sought solution to a drawing of the reduction graph G with at most k crossings; indeed, the edge e_j arrives to the right side of the frame by $n+1$ levels higher that its beginning on the left side. ◀

▶ **Lemma 3.12.** *If G admits a drawing $\mathcal{G}$ such that $\mathrm{cr}(\mathcal{G}) \leq k$, then the original* Satisfiability *instance $(\mathcal{C}, \mathcal{V})$ is satisfiable.*

Proof. If $\mathrm{cr}(\mathcal{G}) \leq k$, then $\mathcal{G}$ satisfies the drawings claims of Section 3.4 and, in particular, Theorem 3.9 and Theorem 3.10. We define a valuation $\tau : \mathcal{V} \to \{\texttt{True}, \texttt{False}\}$ as follows; $x_i \in \mathcal{V}$ is set `True` if Var^i pos-side flipped in $\mathcal{G}$ (the `LB-pos` path of Var^i is drawn to the left of the `LB-neg` path of Var^i), and x_i is set `False` otherwise.

Now, for every clause $C_j \in \mathcal{C}$, the edge e_j in $\mathcal{G}$ starts in the down-level of its cells area and ends in the up-level. By Theorem 3.10, there has to be a Var^i-gadget, $i \in [n]$, such that e_j jumps up to the up-level of its cells area within the subdrawing of Var^i in $\mathcal{G}$. By the definition of the cell types in the construction of G, see in Figure 3, this is possible in accordance with Theorem 3.9 (only one crossing with a `C`-edge) only if $x_i \in C_j$ and x_i is set to `True`, or if $\overline{x_i} \in C_j$ and x_i is set to `False`. In other words, if every $C_j \in \mathcal{C}$ is satisfied by our valuation τ. And this completes the proof. ◀

3.6 On Path-width of the Resulting Instance

The last missing ingredient in the proof of Theorem 3.1 (and hence, of Theorem 1.1) is an estimate of the path-width and tree-width of the constructed instance G. To obtain it, we will use the cops-and-robber game characterization from Theorem 2.1.

We start with an auxiliary technical claim.

▶ **Lemma 3.13.** **(*)** *Let H be a graph whose vertex set is partitioned into m disjoint parts $V(H) = A_1 \cup \ldots \cup A_m$, and for some $a_i \leq b_i$, $i \in \{1, \ldots, m\}$, let $A_i = \{v_{i,j} : a_i \leq j \leq b_i\}$. Assume that*

a) *each A_i induces a path in H in the natural order of vertices, i.e. $v_{i,a_i}, v_{i,a_i+1}, \ldots, v_{i,b_i}$;*

b) *if an edge $v_{i,j}v_{i',j'}$ exists in H for $i \neq i'$, then $|i - i'| = 1$ and $|j - j'| \leq 1$; and*

c) *there are no indices $i \neq i', j \neq j'$ such that both $v_{i,j}v_{i',j'} \in E(H)$ and $v_{i,j'}v_{i',j} \in E(H)$.*[1]

Then there exists a valid monotone search strategy for the cop player on H using $m + 1$ cops against an invisible robber. Furthermore, this strategy can be assumed to start with the cop player occupying the vertex subset $\{v_{1,a_1}, \ldots, v_{m,a_m}\}$.

▶ **Proposition 3.14.** *For any given* Satisfiability *instance, the graph G constructed in Section 3.3 (for the proof of Theorem 3.1) is of path-width at most 12 and of tree-width at most 9.*

Proof. The proof would be finished if we find a monotone search strategy for the cop player on G using 13 cops against an invisible robber (implies path-width at most 12), and one using 10 cops against a visible robber (implies tree-width at most 9). We start with the former.

Firstly, let us place 8 cops (see Figure 1) on vertices $r^1_{1,L}$, $r^1_{h+3,L}$, u^{BL}, u^{TL} (left side of G) and $r^n_{1,R}$, $r^n_{h+3,R}$, u^{BR}, u^{TR} (right side of G). Note (see Figure 4) that such a placement of cops separates the set $U_0 \subseteq V(G)$ formed by the vertices of the left and right `HB`-paths of the frame, and of the `R-left` path of Var^1 and the `R-right` path of Var^n from the rest of the graph G. We can now use additional $4 + 1 = 5$ cops to search the subgraph induced by U_0 by using Theorem 3.13. Notice, however, that in this application the levels dealt with in Theorem 3.13 are shifted against the natural indexing from the construction of G.

After the previous initial phase, we continue the search by induction on $i = 1, 2, \ldots, n$. We assume 8 cops placed on vertices $r^i_{1,L}$, $r^i_{h+3,L}$, u^{i-1}_0, w^{i-1}_0 (the latter two being u^{BL}, u^{TL} if $i = 1$) and, again, $r^n_{1,R}$, $r^n_{h+3,R}$, u^{BR}, u^{TR}. Further, we assume that $V(\mathsf{Var}^{i-1})$ (if $i > 1$)

[1] The graph H can be easily pictured as having a planar drawing with the paths on $A_1, \ldots, A_m$ drawn vertically in order from left to right, and other edges joining only neighboring verticals on the same horizontal level or between two consecutive levels.

is already robber-free. We place 4 of the remaining cops onto vertices u_0^i, u_1^i, w_1^i, w_0^i, and subsequently lift the cops from $r^i_{1,L}$, $r^i_{h+3,L}$, u_0^{i-1}, w_0^{i-1}. We now have 5 free cops which can be used to search the `B-/LB-pos` and `B-/LB-neg` paths of Var^i by using Theorem 3.13. Then, we place 2 of the free cops onto $r^{i+1}_{1,L}$, $r^{i+1}_{h+3,L}$, and use the remaining 3 cops to search the `R-right` path of Var^i and the `R-left` path of Var^{i+1}, again by Theorem 3.13. After finishing previous, we may lift the cops from u_1^i and w_1^i, and we are back in the induction assumption with $i+1$ instead of i.

It is easy to verify that the described procedure is a valid monotone search strategy against an invisible robber.

Regarding the tree-width subcase, we use knowledge of robber's position for a slight improvement of the previous search strategy. At the beginning, after placing cops on $r^1_{1,L}$, $r^1_{h+3,L}$, u^{BL}, u^{TL} and $r^n_{1,R}$, $r^n_{h+3,R}$, u^{BR}, u^{TR}, we look at whether the robber is trapped inside the set U_0 and if this is the case, we throw in 9-th cop to catch the robber in U_0 by Theorem 3.13 while using also the 4 cops starting on $r^1_{1,L}$, u^{BL}, $r^n_{1,R}$, u^{BR}.

In the induction phase, whenever the robber is trapped inside the `B-/LB-pos` and `B-/LB-neg` paths of Var^i, we may instead use the 4 cops from $r^n_{1,R}$, $r^n_{h+3,R}$, u^{BR}, u^{TR} to perform the catch by Theorem 3.13. We do likewise in the other subcase of the `R-right` path of Var^i and the `R-left` path of Var^{i+1}. When moving cops from the position on $r^i_{1,L}$, $r^i_{h+3,L}$, u_0^{i-1}, w_0^{i-1} to next u_0^i, u_1^i, w_1^i, w_0^i, we now move cops in pairs – place onto u_0^i, u_1^i and lift from $r^i_{1,L}$, u_0^{i-1}, and then do the other pairs. We observe that the maximum number of cops needed in this strategy is $4+4+2=10$, and this is tight at the moment just before trapping the robber in the `R-right`/`R-left` paths. ◀

4 Conclusion

We have shown that the Crossing Number problem is NP-hard for graphs of path-width 12 (and as a result, even of tree-width 9). It is worth to remark that, since the measures clique-width and rank-width are bounded by $\mathcal{O}(2^{\mathrm{tw}})$, their width decompositions are also too much general to help deal with the Crossing Number problem. On the other hand, there are more restrictive parameterizations worth trying, e.g. treedepth, distance to linear forest (distance to disjoint paths), feedback vertex set number (distance to a forest), or cut-width or bandwidth.

Barely any of the existing results could be extended for the parameters above. Thus, investigation whether any of them could yield fixed-parameter tractability (or W-hardness) of Crossing Number is an interesting venue to explore. However, as it is known that Crossing Number is in FPT when parameterized by the solution value (k) [4,9], it only makes sense to investigate those parameters which do not bound the crossing number itself.

References

1 Therese Biedl, Markus Chimani, Martin Derka, and Petra Mutzel. Crossing number for graphs with bounded pathwidth. *Algorithmica*, 82(2):355–384, 2020. doi:10.1007/S00453-019-00653-X.

2 Sergio Cabello. Hardness of approximation for crossing number. *Discrete Comput. Geom.*, 49(2):348–358, March 2013. doi:10.1007/S00454-012-9440-6.

3 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. doi:10.1137/120872310.

4 Éric Colin de Verdière and Thomas Magnard. An FPT algorithm for the embeddability of graphs into two-dimensional simplicial complexes. In *Proceedings of the 29th European*

Symposium on Algorithms (ESA), pages 32:1–32:17, 2021. See also arXiv:2107.06236. arXiv:2107.06236.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. doi:10.1007/978-3-662-53622-3.

8 Michael R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM J. Algebr. Discrete Methods*, 4(3):312–316, September 1983. doi:10.1137/0604033.

9 Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. doi:10.1016/J.JCSS.2003.07.008.

10 Petr Hliněný. Crossing number is hard for cubic graphs. *J. Comb. Theory, Ser. B*, 96(4):455–471, 2006. doi:10.1016/j.jctb.2005.09.009.

11 Petr Hliněný. Complexity of anchored crossing number and crossing number of almost planar graphs. *CoRR*, abs/2306.03490, 2023. doi:10.48550/arXiv.2306.03490.

12 Petr Hliněný and Gelasio Salazar. On hardness of the joint crossing number. In *ISAAC*, volume 9472 of *Lecture Notes in Computer Science*, pages 603–613. Springer, 2015. doi:10.1007/978-3-662-48971-0_51.

13 Petr Hliněný and Abhisekh Sankaran. Exact crossing number parameterized by vertex cover. In Daniel Archambault and Csaba D. Tóth, editors, *Graph Drawing and Network Visualization - 27th International Symposium, GD 2019, Prague, Czech Republic, September 17-20, 2019, Proceedings*, volume 11904 of *Lecture Notes in Computer Science*, pages 307–319. Springer, 2019. doi:10.1007/978-3-030-35802-0_24.

14 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Stefankovic. Crossing numbers and parameterized complexity. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, pages 31–36. Springer, 2007. doi:10.1007/978-3-540-77537-9_6.

15 Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory B*, 58(1):22–33, 1993. doi:10.1006/jctb.1993.1027.

16 Paul Turán. A note of welcome. *Journal of Graph Theory*, 1(1):7–9, 1977. doi:10.1002/jgt.3190010105.

17 Meirav Zehavi. Parameterized analysis and crossing minimization problems. *Comput. Sci. Rev.*, 45:100490, 2022. doi:10.1016/J.COSREV.2022.100490.

A Polynomial Kernel for Deletion to the Scattered Class of Cliques and Trees

Ashwin Jacob
National Institute of Technology Calicut, Kozhikode, India

Diptapriyo Majumdar
Indraprastha Institute of Information Technology Delhi, New Delhi, India

Meirav Zehavi
Ben-Gurion University of The Negev, Beersheba, Israel

Abstract

The class of graph deletion problems has been extensively studied in theoretical computer science, particularly in the field of parameterized complexity. Recently, a new notion of graph deletion problems was introduced, called *deletion to scattered graph classes*, where after deletion, each connected component of the graph should belong to at least one of the given graph classes. While fixed-parameter algorithms were given for a wide variety of problems, little progress has been made on the kernelization complexity of any of them. Here, we present the first non-trivial polynomial kernel for one such deletion problem, where, after deletion, each connected component should be a clique or a tree - that is, as dense as possible or as sparse as possible (while being connected). We develop a kernel of $\mathcal{O}(k^5)$ vertices for the same.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Kernelization, Scattered Graph Classes, New Expansion Lemma, Cliques or Trees Vertex Deletion

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.41

Related Version *Full Version*: https://arxiv.org/abs/2409.14209

Funding *Diptapriyo Majumdar*: Supported by Science and Engineering Research Board (SERB) grant SRG/2023/001592.
Meirav Zehavi: Supported by European Research Council (ERC) starting grant titled PARAPATH.

1 Introduction

Graph modification problems form one of the most fundamental problem classes in algorithms and graph theory. The input instance of a graph modification problem consists of an undirected/directed graph, and a non-negative integer k, and the objective is to decide if there exists a set of at most k vertices/edges/non-edges whose deletion/addition yields in a graph belonging to some special graph class. A specific graph modification problem allows to perform a specific graph operation, usually being *vertex deletion* or *edge deletion* or *edge addition* or *edge editing*. Each of these operations, and vertex-deletion in particular, have been extensively studied from the perspective of classical and parameterized complexity. For example, some vertex deletion graph problems that have received intense attention in the past three decades include Vertex Cover, Feedback Vertex Set, Cluster Vertex Deletion Set, Interval Vertex Deletion Set, Chordal Vertex Deletion Set, and more (see [3, 6, 7, 8, 9, 14, 17, 20, 32, 28, 38]).

A graph class Π is said to be *hereditary* if Π is closed under taking induced subgraphs. There are several hereditary graph classes Π such that the corresponding Π-Vertex Deletion problem is well-studied. Such examples include all of the problems listed above.

Formally, the Π-Vertex Deletion is defined as follows: given a graph G and a non-negative integer k, we ask whether G contains at most k vertices whose deletion results in a graph belonging to class Π. Lewis and Yannakakis [35] proved that for every non-trivial Π, the Π-Vertex Deletion problem is NP-complete. Later, Cai [5] has proved that if a hereditary graph class Π can be described by a finite set of forbidden subgraphs containing all minimal forbidden subgraphs in the class, then vertex deletion to Π becomes fixed-parameter tractable (FPT).

Most of the computational problems that are NP-hard in general graphs can often be solved in polynomial time when restricted to special graph classes. For example, Vertex Cover is NP-hard on general graphs, but can be solved in polynomial time in forests, bipartite graphs, interval graphs, chordal graphs, claw-free graphs, and bounded treewidth graphs (see [12, 13, 23, 37]). Additionally, several other graph theoretic problems have also been studied in special graph classes (see [2, 4, 11, 18, 22, 24, 30, 31, 33, 36] for some examples). If your input graph is such that each of its connected components belong to one of those special graph classes where the problem is solvable, then the problem can be solved by solving it over each component of the graph. Therefore, such a graph where each of the components belong to different graph classes are interesting. We say that such graphs belong to a scattered graph class. Vertex deletion problems are useful to find a set of few vertices whose removal results in a graph class where the problem of our interest is tractable. Since the same problem is tractable in scattered graph classes (i.e. tractable in each of the graph class), vertex deletion to scattered graph classes are interesting to look at as well.

Many of the graph classes can be characterized by a set of forbidden graphs [15, 34, 23, 10]. Vertex deletion problems for such graph classes boils down to hitting such forbidden subgraphs occuring as induced subgraphs of the input graph. Unlike this, for deletion to a scattered graph class, the deletion set X might separate the vertices of the union of the forbidden subgraphs for each of the graph classes (instead of hitting them) so that all such graphs do not occur in any of the connected components of the graph $G - X$. This ramps up the difficulty for coming up with FPT, approximation and kernelization algorithms for deletion to scattered graph classes. A naive approach of finding the solutions (or kernels) for each of the deletion problems separately and "combining" them is unlikely to work.

Ganian et al. [21] studied backdoors to scattered classes of CSP problems. Subsequently, Jacob et al. [25, 26] built on the works by Ganian et al. [21] and initiated a systematic study of vertex deletion to scattered graph classes. They considered the $(\Pi_1, \dots, \Pi_d)$-Deletion problem where the input instance is a graph G a parameter k with respect to d (constant) hereditary graph classes $\Pi_1, \dots, \Pi_d$. The objective is to decide if there is a set of at most k vertices S such that every connected component of $G - S$ is in Π_i for some $i \in [d]$. After that, Jacob et al. [26] considered specific pairs of hereditary graph classes Π_1 and Π_2 and have provided singly exponential-time fixed-parameter tractable (FPT) algorithms and approximation algorithms for (Π_1, Π_2)-Deletion problems. Very recently, Jansen et al. [29] conducted a follow-up work on $(\Pi_1, \dots, \Pi_d)$-Deletion problems and have improved the results appearing in [25]. A common theme for the FPT algorithms for deletion to scattered graph classes is a non-trivial "unification" of the techniques used in the deletion problems of each of the graph classes.

Our Problem and Results. To the best of our knowledge, vertex deletion to scattered graph classes is essentially unexplored from the perspective of polynomial kernelization that is a central subfield of parameterized complexity. The only folklore result that follows from Jacob et al. [26] states that if there are two hereditary graph classes Π_1 and Π_2 such that both Π_1 and Π_2 can be described by finite forbidden families and P_d (the induced path of d vertices) is a forbidden induced subgraph for Π_1 for some fixed constant d, then the problem (Π_1, Π_2)-Deletion can be formulated as a d-Hitting Set problem and hence admits a polynomial kernel. This folklore result is very restrictive and does not capture any hereditary graph class whose forbidden sets are not bounded by a fixed constant.

In this paper, we initiate the study of vertex deletion to scattered graph classes from the perspective of polynomial kernelization. We consider the problem Cliques or Trees Vertex Deletion where given a graph G and a non-negative integer k, we ask if G contains a set S of at most k vertices, such that $G - S$ is a simple graph and every connected component of it is either a clique or a tree – that is, as dense as possible or as sparse as possible (while being connected). Naturally, we are specifically interested in the case where the input graph is already a simple graph. However, our preprocessing algorithm can produce intermediate multigraphs. Hence, we directly consider this more general formulation. Formally, we define our problem as follows.

Cliques or Trees Vertex Deletion (CTVD) **Parameter:** k
Input: An undirected (multi)graph $G = (V, E)$ and a non-negative integer k.
Question: Does G contain a set S of at most k vertices such that $G - S$ is a simple graph and every connected component of $G - S$ is either a clique or a tree?

This problem is particularly noteworthy as it captures the essence of scattered classes: allowing the connected components to belong to vastly different graph classes and ideally the simplest ones where various computational problems are polynomial-time solvable. Here, we indeed consider the extremes: the simplest densest graph (cliques) and the most natural class of sparsest connected graphs (trees). If X is a feasible solution to Cliques or Trees Vertex Deletion for a graph G, then we call X a *(clique, tree)-deletion set* of G. We consider the (upper bound on the) solution size k as the most natural parameter. Jacob et al. [26] proved that Cliques or Trees Vertex Deletion is in FPT - specifically, that it admits an algorithm that runs in $\mathcal{O}^*(4^k)$-time. In this paper, we prove the following result on the polynomial kernelization for this problem.

▶ **Theorem 1.1.** Cliques or Trees Vertex Deletion *(CTVD) admits a kernel with* $\mathcal{O}(k^5)$ *vertices.*

Our theorem is the first non-trivial result on a polynomial kernel for vertex deletion to pairs of graph classes. The proof of this kernelization upper bound is based on several non-trivial insights, problem specific reduction rules, and structural properties of the solutions.

Organization of the Paper. In Section 2, we introduce basic terminologies and notations. Section 3 is devoted to the proof of our main result (Theorem 1.1). Finally, in Section 4, we conclude with some future research directions.

2 Preliminaries

We use standard graph theoretic terminologies from Diestel's book [15]. For a vertex v in G, let $d_G(v)$ denote the degree of v in G, which is the number of edges in G incident to v. When we look at the number of edges incident on v, we take the multiplicity of every edge into account. A *pendant vertex* in a graph G is a vertex having degree one in G. A *pendant edge* in a graph G is the edge incident to a pendant vertex in G. A *path* P in a graph is a sequence of distinct vertices $(v_1, \ldots, v_r)$ such that for every $1 \leq i \leq r-1$, $v_i v_{i+1}$ is an edge.

A *degree-2-path* in G is a path P such that all its internal vertices have degree exactly 2 in G. If a graph G has a degree-2-path $P = (v_1, \dots, v_r)$ such that v_1 is a pendant vertex, for every $i \in \{2, \dots, r-1\}$, $d_G(v_i) = 2$ and $d_G(v_r) > 2$, then we call P a *degree-2-tail of length* r. If G has a degree-2-path $P = (v_1, \dots, v_r)$ such that for all $i \in \{2, \dots, r-1\}$, $d_G(v_i) = 2$, and $d_G(v_1), d_G(v_r) > 2$, then we call P a *degree-2-overbridge of length* r. Sometimes, for simplicity, we use $P = v_1 - v_2 - \dots - v_r$ to denote the same path P of r vertices. The graph K_t for integer $t \geq 1$ is the clique of t vertices. For a non-negative integer c, the graph cK_1 is the collection of c isolated vertices. The graphs C_t and P_t for integer $t \geq 1$ are the cycle and path of t vertices respectively. We define a *paw graph* as the graph with four vertices u_1, u_2, u_3 and u_4 where u_1, u_2, u_3 form a triangle, and u_1 alone is adjacent to u_4 (thus, u_4 is a pendant vertex). We define a *diamond graph* as the graph with four vertices u_1, u_2, u_3 and u_4 where u_1, u_2, u_3 form a triangle, and u_1, u_2 are adjacent to u_4. Note that both paw and diamond contain a triangle as well as a $2K_1$ as induced subgraphs. We say that a vertex $x \in V(G)$ is *adjacent* to a subgraph $G[Y]$ for some $x \notin Y$ if x has a neighbor in Y in the graph G. A *cut* of G is a bipartition (X, Y) of $V(G)$ into nonempty subsets X and Y. The set $E_G(X, Y)$ is denoted as the edges *crossing the cut.* We omit the subscript when the graph is clear from the context. Let C be a cycle having r vertices We use $C = v_1 - v_2 - \dots - v_r - v_1$ to denote the cycle with edges $v_i v_{i+1}$ for every $1 \leq i \leq r-1$ and the edge $v_r v_1$.

▶ **Definition 2.1** (v-flower). *For a graph G and a vertex v in G, a* v-flower *is the structure formed by a family of ℓ cycles $C_1, C_2, \dots C_\ell$ in G all containing v and no two distinct cycles C_i and C_j sharing any vertex except v. We refer to the C_is' as the* petals *and to v as the* core*. The number of cycles ℓ is the* order *of the v-flower.*

▶ **Proposition 2.2** ([13], Lemma 9.6). *Given a graph G with $v \in V(G)$ and an integer k, there exists a polynomial-time algorithm that either provides a v-flower of order $k+1$ or compute a set $Z \subseteq V(G) \setminus \{v\}$ with at most $2k$ vertices satisfying the following properties: Z intersects every cycle of G that passes through v, and there are at most $2k$ edges incident to v and with second endpoint in Z.*

Let q be a positive integer and G be a bipartite graph with vertex bipartition (A, B). For $\widehat{A} \subseteq A$ and $\widehat{B} \subseteq B$, a set $M \subseteq E(G)$ of edges is called a *q-expansion* of $\widehat{A}$ into $\widehat{B}$ if

(i) every vertex of $\widehat{A}$ is incident to exactly q edges of M, and
(ii) exactly $q|\widehat{A}|$ vertices of $\widehat{B}$ are incident to the edges in M.

The vertices of $\widehat{A}$ and $\widehat{B}$ that are the endpoints of the edges of M are said to be *saturated* by the q-expansion M. We would like to clarify that by definition of q-expansion M of $\widehat{A}$ into $\widehat{B}$, all vertices of $\widehat{A}$ are saturated by M, and $|\widehat{B}| \geq q|\widehat{A}|$. But not all vertices of $\widehat{B}$ are guaranteed to be saturated by M.

▶ **Lemma 2.3** (q-Expansion Lemma [39, 13]). *Let $q \in \mathbb{N}$ and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$, and there is no isolated vertex in B. Then, there exist non-empty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that*

(i) *there is a q-expansion M of X into Y, and*
(ii) *no vertex in Y has a neighbor outside X, that is, $N(Y) \subseteq X$.*

Furthermore, the sets X and Y can be found in time $\mathcal{O}(mn^{1.5})$.

Recently, Fomin et al. [19] have designed the following generalization of the Lemma 2.3 (q-Expansion Lemma) as follows.

▶ **Lemma 2.4** (New q-Expansion Lemma [19, 1, 27]). *Let q be a positive integer and G be a bipartite graph with bipartition (A, B). Then there exists $\widehat{A} \subseteq A$ and $\widehat{B} \subseteq B$ such that there is a q-expansion M of $\widehat{A}$ into $\widehat{B}$ in G such that*

(i) *$N(\widehat{B}) \subseteq \widehat{A}$, and*
(ii) $|B \setminus \widehat{B}| \leq q|A \setminus \widehat{A}|$.
Furthermore, the sets $\widehat{A}$, $\widehat{B}$ and the q-expansion M can be computed in polynomial-time.

Observe that the Lemma 2.4 statement does not require the two conditions that B has no isolated vertex and $|B| \geq q|A|$ that were required for the Lemma 2.3. In particular, if $|B| > q|A|$, then it must be that $|\widehat{B}| > q|\widehat{A}|$ and $\widehat{B}$ will contain some vertex that is not saturated by the q-expansion M.

Forbidden Subgraph Characterization. Given a graph class $\mathcal{G}$, any (induced) subgraph that is not allowed to appear in any graph of $\mathcal{G}$ is called an *obstruction* for $\mathcal{G}$ (also known as forbidden subgraphs or forbidden induced subgraphs). We first identify the obstructions for Cliques or Trees Vertex Deletion. Clearly, on simple graphs, we cannot have an obstruction for both a tree and a clique in the same connected component. If $\mathcal{G}$ is the class of all cliques, then the obstruction for $\mathcal{G}$ is $2K_1$ and if $\mathcal{G}$ is the class of all forests, then any cycle C_t with $t \geq 3$ is an obstruction for $\mathcal{G}$. Note that a cycle C_t with $t \geq 4$ contains $2K_1$ as an induced subgraph. Throughout the paper, we sometimes abuse the notation where an obstruction (or a forbidden induced subgraph) is viewed as a set and sometimes it is viewed as an (induced) subgraph.

▶ **Observation 2.5.** *For every integer $t \geq 4$, the cycles C_t contains $2K_1$ as induced subgraph.*

Thus, we can conclude that the obstructions for Cliques or Trees Vertex Deletion are cycles C_t with $t \geq 4$ and connected graphs with both $2K_1$ and C_3 as induced subgraphs. For multigraphs, a vertex with a self-loop and two vertices with two (or more) edges are obstructions as well. If a connected graph has both $2K_1$ and C_3 as induced subgraphs, a (clique, tree)-deletion set either intersects the union of the vertex sets of these subgraphs or contains a subset separating them. The following lemma claims that if a connected graph contains both $2K_1$ and C_3 as induced subgraphs, then it contains a paw or a diamond.

▶ **Lemma 2.6** ($\star$).[1] *A connected graph G with both $2K_1$ and C_3 as induced subgraphs contains either a paw or a diamond as an induced subgraph.*

From Lemma 2.6, we get a forbidden subgraph characterization for the class of graphs where each connected component is a clique or a tree.

▶ **Lemma 2.7** ($\star$). *Let $\mathcal{G}$ be the class of all simple graphs where each connected component is a clique or a tree. Then, a simple graph G belongs to $\mathcal{G}$ if and only if G does not contain any paw, diamond or cycle C_i with $i \geq 4$ as an induced subgraph.*

Parameterized Complexity and Kernelization. A *parameterized problem L* is a set of instances $(x, k) \in \Sigma^* \times \mathbb{N}$ where Σ is a finite alphabet and $k \in \mathbb{N}$ is a parameter. The notion of "tractability" in parameterized complexity is defined as follows.

▶ **Definition 2.8** (Fixed-Parameter Tractability). *A parameterized problem L is said to be* fixed-parameter tractable *(or FPT) if given $(x, k) \in \Sigma^* \times \mathbb{N}$, there is an algorithm $\mathcal{A}$ that correctly decides if $(x, k) \in L$ in $f(k)|x|^{\mathcal{O}(1)}$-time for some computable function $f : \mathbb{N} \to \mathbb{N}$. This algorithm $\mathcal{A}$ is called* fixed-parameter algorithm *(or* FPT algorithm*) for the problem L.*

[1] Due to lack of space, the proofs that are omitted or marked $\star$ can be found in the full version.

Observe in the above definition that we allow combinatorial explosion with respect to the parameter k while the algorithm runs in polynomial-time with respect to $|x|$. We say that two instances (x, k) of L and (x', k') of L are *equivalent* if $(x, k) \in L$ if and only if $(x', k') \in L$. The notion of kernelization (also known as parameterized preprocessing) is defined as follows.

▶ **Definition 2.9** (Kernelization). *A* kernelization *for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given an instance (x, k) of L, outputs an equivalent instance (x', k') (called* kernel*) of L in time polynomial in $|x| + k$ such that $|x'| + k' \leq g(k)$ for some function $g : \mathbb{N} \to \mathbb{N}$. If $g(k)$ is $k^{\mathcal{O}(1)}$, then L is said to admit a* polynomial kernel.

A kernelization algorithm usually consists of a collection of *reduction rules* that have to be applied exhaustively in sequence. A reduction rule is *safe* if given an instance (x, k) of L, one application of the reduction rule outputs an equivalent instance (x', k') of L. It is well known that "a decidable parameterized problem is FPT if and only if it admits a kernelization". For more details, we refer to [13, 16, 19] for more formal definitions about parameterized complexity and kernelization.

3 A Polynomial Kernel for Cliques and Trees

This section is devoted to a polynomial kernel for Cliques or Trees Vertex Deletion. As the first step, we invoke the following proposition by Jacob et al. [26] that computes a (clique, tree)-deletion set $S \subseteq V(G)$ with at most $4k$ vertices.

▶ **Proposition 3.1** ([26], Theorem 6). Cliques or Trees Vertex Deletion *admits a 4-approximation algorithm.*

We begin with the following observation, whose proof is trivial.

▶ **Observation 3.2.** *For any subset $Z \subseteq V(G)$, if (G, k) is a yes-instance for* Cliques or Trees Vertex Deletion *with solution X, then $(G - Z, k)$ is a yes-instance for* Cliques or Trees Vertex Deletion *with solution $X \setminus Z$.*

Overview of the Kernelization Algorithm. We start with invoking the 4-approximation algorithm for Cliques or Trees Vertex Deletion(Proposition 3.1) to get a (clique, tree)-deletion set S of size at most $4k$. In Section 3.1, we provide some reduction rules that guarantee that every connected component of $G - S$ has a neighbor in S, and the graph has no degree-2-path of G has more than four vertices. Subsequently, in Section 3.2, we provide some reduction rules and prove that the number of vertices in the connected components of $G - S$ that are cliques is $\mathcal{O}(k^5)$. Finally, in Section 3.3, we reduce the number of vertices in the connected components of $G - S$ that are trees to $\mathcal{O}(k^2)$. For this, we prove a variant of Proposition 2.2 and use reduction rules related to that using New q-Expansion Lemma (i.e. Lemma 2.4).

3.1 Initial Preprocessing Rules

Let S be a 4-approximate (clique, tree)-deletion set of G obtained from Proposition 3.1. If $|S| > 4k$, we conclude that (G, k) is a no-instance and return a trivial constant sized no-instance. Hence, we can assume without loss of generality that $|S| \leq 4k$. We also assume without loss of generality that G has no connected component that is a clique or a tree. If such components are there, we can delete those components. Hence, we can naturally assume from now onwards, that every connected component of $G - S$ (that is either a clique or a tree) has some neighbor in S. But some of our subsequent reduction rules can create some component in $G - S$ that is neither a clique nor a tree. So, we state the following reduction rule for the sake of completeness. In this following reduction rule, we delete isolated connected components C in $G - S$, whose safeness easily follows. All the obstructions for Cliques or Trees Vertex Deletion are connected graphs and intersect with S. Thus, no obstruction can be part of isolated component C, and also S.

▶ **Reduction Rule 3.3.** *If there exists a connected component C in $G - S$ such that no vertex in C has a neighbor in S, then remove C from G. The new instance is $(G - C, k)$.*

Since one of our subsequent reduction rules can create parallel edges. As parallel edges are also obstructions, we state the following reduction rule whose safeness is also trivial.

▶ **Reduction Rule 3.4.** *If there is an edge with multiplicity more than two, reduce the multiplicity of that edge to exactly two.*

We also have the following reduction rule that helps us to bound the number of pendant vertices attached to any vertex.

▶ **Reduction Rule 3.5.** *If there exists a vertex u in G adjacent to vertices v and v' that are pendants in G, then remove v from G. The new instance is $(G - v, k)$.*

▶ **Lemma 3.6** ($\star$)**.** *Reduction Rule 3.5 is safe.*

We can conclude that if Reduction Rule 3.5 is not applicable, then every vertex in G is adjacent to at most one pendant vertex in G. From now, we assume that every vertex in G is adjacent to at most one pendant vertex. Our next two reduction rules help us to reduce the length (the number of vertices) of a degree-2-path in G. Note that a degree-2-path can be of two types, either a degree-2-tail or a degree-2-overbridge. The following two reduction rules handle both such types.

▶ **Reduction Rule 3.7.** *Let $P = (v_1, v_2, \dots, v_\ell)$ be degree-2-tail of length ℓ such that $d_G(v_1) > 2$, $d_G(v_\ell) = 1$ and $Z = \{v_3, v_4, \dots, v_\ell\}$. Then, remove Z from G. The new instance is $(G - Z, k)$.*

▶ **Lemma 3.8** ($\star$)**.** *Reduction Rule 3.7 is safe.*

Our previous reduction rule has illustrated that we can shorten a long degree-2-tail to length at most two. Now, we consider a degree-2-overbridge P of length ℓ. Our next lemma gives us a structural characterization that if we delete all but a few vertices of P, then the set of all paws and diamonds remain the same even after deleting those vertices.

▶ **Lemma 3.9** ($\star$)**.** *Let $P = (v_1, v_2, \dots, v_\ell)$ be a degree-2-overbridge of length ℓ in G and $Z = \{v_3, v_4, \dots, v_{\ell-2}\}$. Consider the graph G' obtained from G by deleting the vertices of Z and then adding the edge $v_2v_{\ell-1}$. Then the following statements hold true.*

(i) *Every paw and every diamond of G is disjoint from Z.*
(ii) *Every paw and every diamond of G' is disjoint from $\{v_2, v_{\ell-1}\}$.*
(iii) *The set of paws and diamonds in both G and $G - Z$ are the same.*

Our next reduction rule exploits the above lemma and reduces the length of a degree-2-overbridge to at most four.

▶ **Reduction Rule 3.10.** *Let $P = (v_1, v_2, \dots, v_\ell)$ be a degree-2-overbridge of length ℓ in G and $Z = \{v_3, v_4, \dots, v_{\ell-2}\}$. Let G' be the graph obtained from G by removing Z and adding the edge $v_2v_{\ell-1}$. The new instance is (G', k). We refer to Figure 1 for an illustration.*

▶ **Lemma 3.11** ($\star$)**.** *Reduction Rule 3.10 is safe.*

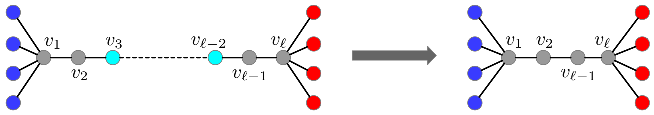


Figure 1 An illustration of applying Reduction Rule 3.10.

3.2 Bounding the Clique Vertices in $G - S$

Let $V_1 \subseteq V(G) \setminus S$ denote the set of vertices of the connected components of $G - S$ that form cliques of size at least 3. We now bound the number of connected components in $G[V_1]$ (which are cliques). Let us create an auxiliary bipartite graph $H = (S, \mathcal{C})$ with S on one side and $\mathcal{C}$ having a vertex set corresponding to each of the clique connected components in V_1 on the other side. We add an edge (s, C) with $s \in S$ and $C \in \mathcal{C}$ if s is adjacent to at least one vertex in C. We now show how to ensure that $|\mathcal{C}| \leq 2|S|$. Note that by Reduction Rule 3.3, no component in $\mathcal{C}$ is an isolated vertex in H. So, we have the following reduction rule, where we rely on the Expansion Lemma.

▶ **Reduction Rule 3.12.** *If $|\mathcal{C}| \geq 2|S|$, then call the algorithm provided by the q-Expansion Lemma with $q = 2$ (Lemma 2.3) to compute sets $X \subseteq S$ and $\mathcal{Y} \subseteq \mathcal{C}$ such that there is a 2-expansion M of X into $\mathcal{Y}$ in H and $N_H(\mathcal{Y}) \subseteq X$. The new instance is $(G - X, k - |X|)$.*

▶ **Lemma 3.13** ($\star$). *Reduction Rule 3.12 is safe.*

Thus, we have the following observation.

▶ **Observation 3.14.** *After exhaustive applications of Reduction Rules 3.3- 3.12, $|\mathcal{C}| \leq 8k$.*

We now give one of the most crucial reduction rules that gives us an upper bound the size of every clique in $G[V_1]$. We have the following marking scheme for each of the cliques in $G[V_1]$.

Procedure 1 Mark-Clique-K.

For every non-empty subset Z of size at most 3 of S, for every function $f : Z \to \{0, 1\}$, let $K_{Z,f}$ be the set of vertices v in K such that for each $z \in Z$,
- if $f(z) = 1$, then v is adjacent to z.
- if $f(z) = 0$, then v is not adjacent to z.

We arbitrarily mark $\min\{|K_{Z,f}|, k + 4\}$ vertices of $K_{Z,f}$. Note that we have marked at most $\varepsilon(k) = (2^3\binom{4k}{3} + 2^2\binom{4k}{2} + 2\binom{4k}{1})(k + 4)$ vertices in K. Let $v \in K$ be a vertex that is not marked by the above procedure Mark-Clique-K. The following set of lemmas illustrate that v is an irrelevant vertex of G.

▶ **Lemma 3.15** ($\star$). *Let S be a (clique, tree)-deletion set of at most $4k$ vertices and K be a connected component of $G - S$ that is a clique. Moreover, let $v \in K$ be a vertex that is not marked by the procedure* Mark-Clique-K *and $X \subseteq V(G) \setminus \{v\}$ be a set of at most k vertices. If $G - X$ has a vertex subset O and $G[O]$ is isomorphic to C_4, or a diamond or a paw then $G - (X \cup \{v\})$ also contains a C_4, or a diamond, or a paw as an induced subgraph.*

Our previous lemma has illustrated that if $G - X$ has a paw or a diamond or a C_4 as an induced subgraph, then $G - (X \cup \{v\})$ also has a paw or a diamond or a C_4 respectively. We will now illustrate and prove an analogous statement when $G - X$ has an induced cycle of length larger than 4. We begin with the following observation.

▶ **Observation 3.16** (⋆). *Let $C = v - u - u_1 - u_2 - \ldots - u' - v$ be a cycle of length at least 5 in G where the path $P = u - u_1 - u_2 - \ldots - u'$ is an induced path in G. Then there exists a cycle of length at least 4 or a diamond as an induced subgraph in G.*

Using the above observation, we can prove the following lemma.

▶ **Lemma 3.17.** *Let S be a (clique, tree)-deletion set set of at most $4k$ vertices and K be a connected component of $G - S$ that is a clique. Moreover, let $v \in K$ be a vertex that is not marked by the procedure* Mark-Clique-K *and $X \subseteq V(G) \setminus \{v\}$ be a set of at most k vertices. If $G - X$ has an induced cycle of length at least 5, then there exists a cycle of length at least 4 or a diamond as an induced subgraph in $G - (X \cup \{v\})$.*

Proof. Suppose that the premise of the statement is true but for the sake of contradiction, we assume that $G - (X \cup \{v\})$ does not have cycles of length at least 4 and diamonds as induced subgraphs. Since v is the only vertex that is in G but not in $G - \{v\}$, it follows that there is an induced cycle C of length at least 5 in $G - X$ such that $v \in C$. Note that C has at most two vertices from K including v as K is a clique. Furthermore, it must have two non-adjacent vertices from S as otherwise C contains a triangle or an induced C_4, contradicting that C is an induced cycle of length at least 5. Let $z_1, z_2 \in C \cap S$ that are non-adjacent. There are two cases.

Case (i): The first case is $|C \cap K| = 1$ and let $C \cap K = \{v\}$. Then, both z_1 and z_2 are adjacent to $v \in C$. Let us define a function $f : \{z_1, z_2\} \to \{0, 1\}$ with $f(z_1) = 1$ and $f(z_2) = 1$. For the set $\{z_1, z_2\}$ and the function f, the vertex v is unmarked by the procedure Mark-Clique-K. Hence, there are $k+4$ vertices that are adjacent to both z_1 and z_2 and are marked by the procedure. All the marked vertices are in $K \setminus \{v\}$ out of which at most k vertices are in X. Hence, there is $v' \in K \setminus X$ such that v' is adjacent to both z_1 and z_2 and is marked by the procedure. Let us look at the cycle $C' = (C \setminus \{v\}) \cup \{v'\}$. Note that C' is a cycle where the path from z_1 to z_2 is an induced path in $G - X$. There could be edges from v' to other vertices of C'. By Observation 3.16, C' is either an induced cycle of length at least 4 in G or it induces a diamond. Since $C' \cap (X \cup \{v\}) = \emptyset$, this contradicts our initial assumption that $G - (X \cup \{v\})$ does not have cycles of length at least 4 and diamonds as induced subgraphs.

Case (ii): The second and last case is $|C \cap K| = 2$. Let $v, x \in C \cap K$. Observe that v and x are two consecutive vertices in C. Since $vx \in E(G)$, it must be that v is adjacent to z_1 and x is adjacent to z_2. As C is an induced cycle of length at least 5, it must be that z_1 is not adjacent to x and z_2 is not adjacent to v.

Let us define a function $f : \{z_1, z_2\} \to \{0, 1\}$ with $f(z_1) = 1$ and $f(z_2) = 0$. For the set $\{z_1, z_2\}$ and the function f, the vertex v is unmarked by the procedure Mark-Clique-K. Hence, there are $k + 4$ vertices that is adjacent to z_1 and not adjacent to z_2 that are marked by the procedure. All the marked vertices are in $K \setminus \{v\}$ out of which at most k vertices are in X. Hence, there is $v' \in K \setminus X$ such that v' is adjacent to z_1 and not adjacent to z_2 that is marked by the procedure.

We replace v in C by v' to get a new cycle C' that has the same number of vertices as C (see Figure 2 for an illustration). Note that C' is a cycle where the path from z_1 to x is an induced path in $G - X$. There could be edges from v' to other vertices of C'. By Observation 3.16, C' is either an induced cycle of length at least 4 in G or it induces a diamond. This contradicts our initial assumption that $G - (X \cup \{v\})$ does not have cycles of length at least 4 and diamonds as induced subgraphs.

Since the above cases are mutually exhaustive, this completes the proof. ◀

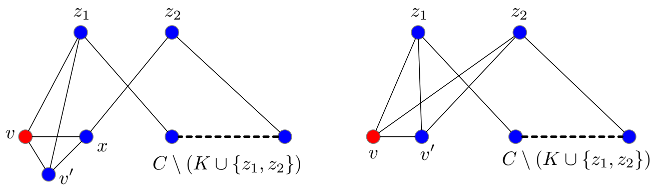

z_1
z_2
v
x
v'
$C \setminus (K \cup \{z_1, z_2\})$
z_1
z_2
v
v'
$C \setminus (K \cup \{z_1, z_2\})$

Figure 2 An illustration of C_5.

Consider a connected component K of $G - S$ such that S is a (clique, tree)-deletion set of G with at most $4k$ vertices and K is a clique. Lemma 3.15 and Lemma 3.17 illustrate that if a vertex $v \in K$ is not marked by the procedure Mark-Clique-K, then deleting v from the graph is safe. As a consequence of this, we have the following reduction rule the safeness of which follows from the above two lemmas.

▶ **Reduction Rule 3.18.** *Let K be a connected component in $G[V_1]$. If $v \in K$ is an unmarked vertex after invoking the procedure* Mark-Clique-K*, then remove v from G. The new instance is $(G - v, k)$.*

▶ **Lemma 3.19.** *Reduction Rule 3.18 is safe.*

Proof. The forward direction ($\Rightarrow$) is trivial as if X is a solution of size at most k in G, then by Observation 3.2, $X \setminus \{v\}$ is a solution of size at most k in $G - v$.

For the backward direction ($\Leftarrow$), let X be a solution of size at most k in $G - v$. Targeting a contradiction, suppose X is not a solution in G. Then, there exists an obstruction O of Cliques or Trees Vertex Deletion in $G - X$ containing v that is a diamond, or a paw, or C_i where $i \geq 4$ due to Lemma 2.7. If O is isomorphic to a C_4, or a diamond, or a paw, then by Lemma 3.15, $G - (X \cup \{v\})$ also contains a C_4, or a diamond, or a paw as induced subgraph. It contradicts Lemma 2.7 that X is a solution to $G - \{v\}$. Else, O is isomorphic to C_i where $i \geq 5$. By Lemma 3.17, $G - (X \cup \{v\})$ contains a C_j, where $j \geq 4$, or a diamond as induced subgraph. This contradicts Lemma 2.7 as X is a solution to $G - \{v\}$. Hence X is a solution to G. ◀

We have the following lemma that bounds $|V_1|$, i.e. the number of vertices that are part of cliques of size at least 3 in $G - S$.

▶ **Lemma 3.20.** *Let G be the graph obtained after exhaustive application of Reduction Rules 3.3 to 3.18. Then $|V_1| \leq 8k\varepsilon(k)$ where $\varepsilon(k) = (k+4)(8\binom{4k}{3} + 4\binom{4k}{2} + 2\binom{4k}{1})$.*

Proof. Since Reduction Rules 3.3-3.12 are not applicable, it follows from Observation 3.14 that the number of connected components in $G[V_1]$ is at most $8k$. Every connected component of $G[V_1]$ is a clique. For every connected component K of $G[V_1]$, observe that the procedure Mark-Clique-K marks at most $\varepsilon(k)$ vertices from K. Since Reduction Rule 3.18 is not applicable, $G[V_1]$ has no unmarked vertices. As $G[V_1]$ has at most $8k$ connected components, it follows that $|V_1| \leq 8k\varepsilon(k)$. ◀

3.3 Bounding the Tree Vertices in $G - S$

In this section, we describe the set of reduction rules that we use to reduce the number of vertices that participate in the forests of $G - S$. Let $V_2 = V \setminus (S \cup V_1)$. Note that $G[V_2]$ is the collection of trees in $G - S$.

▶ **Reduction Rule 3.21.** *Let v be a leaf in a connected component C of $G[V_2]$ such that neither v nor its neighbor in C is adjacent to any vertex of S. Then, delete v from G and the new instance is $(G - v, k')$ with $k' = k$.*

▶ **Lemma 3.22** ($\star$)**.** *Reduction Rule 3.21 is safe.*

Let C be a connected component of $G[V_2]$. We say that C is a *pendant tree* of $G[V_2]$ if either C consists of pendant vertex of G or has a unique vertex u that has a unique neighbour in S and no other vertex of C has any neighbor in S. We have the following observation.

▶ **Observation 3.23** ($\star$)**.** *Let C be a connected component of $G[V_2]$. Then, C is a pendant tree if and only if $E(C, V(G) \setminus C)$ contains a single edge.*

Given a pendant tree C of $G[V_2]$, we call $x \in S$ the *unique S-neighbor* of C if $N_G(C) = \{x\}$ and $|N_G(x) \cap C| = 1$. Furthermore, given a vertex $x \in S$, we call C a *pendant tree-neighbor* of x if C is a pendant tree of $G[V_2]$ and x is the *unique S-neighbor* of C.

▶ **Reduction Rule 3.24.** *Let C be a pendant tree in $G[V_2]$ such that $x \in S$ is a unique S-neighbor of C. Then, delete all the vertices of C except for the vertex u that has the unique S-neighbor $x \in S$ in C to obtain the graph G'. Let (G', k') be the output instance such that $k' = k$.*

▶ **Lemma 3.25** ($\star$)**.** *Reduction Rule 3.24 is safe.*

▶ **Lemma 3.26** ($\star$)**.** *If Reduction Rule 3.24 is not applicable to the input instance (G, k), then any pendant tree of $G[V_2]$ is a pendant vertex of G.*

Recall from Definition 2.1 that a v-flower of order r is a collection of r cycles that pairwise intersect at v and are pairwise disjoint otherwise. Our next reduction rule uses the concept of v-flower and Proposition 2.2 as follows.

▶ **Reduction Rule 3.27.** *For $v \in S$, we invoke Proposition 2.2 in $G[V_2 \cup \{v\}]$. If this gives a v-flower of order $3k + 2$, then delete v from G and the new instance is $(G - v, k - 1)$.*

▶ **Lemma 3.28** ($\star$)**.** *Reduction Rule 3.27 is safe.*

Since Reduction Rule 3.27 is not applicable, invoking Proposition 2.2 of order $(3k + 2)$ at $G[\{v\} \cup V_2]$ gives us a set $H_v \subseteq V_2$ of at most $6k + 4$ vertices that intersects all cycles of $G[\{v\} \cup V_2]$ that passes through v. Our following lemma proves that the same vertex subset H_v also intersects all paws and diamonds of $G[\{v\} \cup V_2]$ passing through v.

▶ **Lemma 3.29** ($\star$)**.** *If Reduction Rule 3.27 is not applicable, then polynomial time, we can obtain a vertex subset $H_v \subseteq V_2$ with $|H_v| \leq 6k + 4$ such that H_v intersects every cycle, every paw, and every diamond in $G[\{v\} \cup V_2]$ that passes through v.*

When the above mentioned reduction rules are not applicable, we have the following lemma which bounds the number of connected components of the graph $G[V_2 \setminus H_v]$ that is adjacent to only v.

Construction of Auxiliary Bipartite Graph. If none of the above reduction rules are applicable, we exploit the structural properties of the graph using the above mentioned lemmas and construct an auxiliary bipartite graph that we use in some reduction rules later. Let $\mathcal{C}$ denote the connected components of $G[V_2 \setminus (H_v \cup \{v\})]$ that are adjacent to v. In other words, if $D \in \mathcal{C}$, then D is a connected component of $G[V_2 \setminus (H_v \cup \{v\})]$ such that v is adjacent to D.

▶ **Definition 3.30.** *Given $v \in S$, let H_v denote the set of at most $6k + 4$ vertices obtained by Lemma 3.29 for the graph $G[\{v\} \cup V_2]$. Consider the graph $G[V_2 \setminus (H_v \cup \{v\})]$ that is a forest. We define an auxiliary bipartite graph $\mathcal{H} = (H_v \cup (S \setminus \{v\}), \mathcal{C})$ where $H_v \cup (S \setminus \{v\})$ is on one side, and $\mathcal{C}$ on the other side. The set $\mathcal{C}$ contains a vertex for each connected component C of $G[V_2 \setminus (H_v \cup \{v\})]$ that has a vertex adjacent to v. We add an edge between $h \in H_v \cup (S \setminus \{v\})$, and connected component $C \in \mathcal{C}$ if h is adjacent to a vertex in component $C \in \mathcal{C}$.*

We prove the following observation that is crucial to the next reduction rule which reduces the number of edges incident to a vertex $v \in S$ with other endpoint being V_2.

▶ **Observation 3.31 ($\star$).** *Let $\mathcal{H} = (H_v \cup (S \setminus \{v\}), \mathcal{C})$ be the auxiliary bipartite graph as defined in Definition 3.30. If v has degree more than $60(k + 1)$ in $G[\{v\} \cup V_2]$, then $\mathcal{C}$ has more than $4(|S| + |H_v|)$ components.*

Applying the New Expansion Lemma. If there is a vertex $v \in S$ such that there are at least $60(k + 1)$ edges incident to v with the other endpoints being in V_2, then Observation 3.31 implies that $|\mathcal{C}| > 4(|S| + |H_v|)$. Suppose that we apply new 4-expansion lemma (Lemma 2.4 with $q = 4$) on $\mathcal{H}$ to obtain $A \subseteq (S \setminus \{v\}) \cup H_v, \mathcal{B} \subseteq \mathcal{C}$ with a 4-expansion $\widehat{M}$ of A into $\mathcal{B}$. Then, it satisfies that (i) $|\mathcal{C} \setminus \mathcal{B}| \leq 4|(S \cup H_v) \setminus A|$ and $N_{\mathcal{H}}(\mathcal{B}) \subseteq A$. As $|\mathcal{C} \setminus \mathcal{B}| \leq 4|(S \cup H_v) \setminus A|$ and $|\mathcal{C}| > 4(|S| + |H_v|)$, it must be that $|\mathcal{B}| > 4|A|$. Then, there must be a component $C^* \in \mathcal{B}$ such that C^* is not an endpoint of $\widehat{M}$ (or not saturated by $\widehat{M}$). Let $\widehat{\mathcal{B}} \subseteq \mathcal{B}$ denote the components of $\mathcal{B}$ that are saturated by $\widehat{M}$. As some component of $\mathcal{B}$ is not in $\widehat{\mathcal{B}}$, it must be that $\widehat{\mathcal{B}} \subset \mathcal{B}$. We use these characteristics crucially to prove that our next reduction rule is safe.

▶ **Reduction Rule 3.32.** *Let $v \in S$ be a vertex with degree at least $60(k + 1)$ in $G[\{v\} \cup V_2]$ and let $\mathcal{H}$ be the auxiliary bipartite graph as illustrated in Definition 3.30. We invoke the algorithm provided by Lemma 2.4 (i.e. new q-expansion lemma with $q = 4$) to compute sets $A \subseteq H_v \cup (S \setminus \{v\})$ and $\mathcal{B} \subseteq \mathcal{C}$ such that A has a 4-expansion $\widehat{M}$ into $\mathcal{B}$ in $\mathcal{H}$ and $N_{\mathcal{H}}(\mathcal{B}) \subseteq A$. Let $\widehat{\mathcal{B}} \subseteq \mathcal{B}$ denotes the vertices of $\mathcal{B}$ that are saturated by $\widehat{M}$ (endpoints of $\widehat{M}$ in $\mathcal{B}$). Remove the edges between v and the connected components in $\widehat{\mathcal{B}}$ in G and create a double edge between v and every vertex in A to obtain the graph G'. The new instance is (G', k') with $k = k'$. We refer to the Figure 3 for an illustration.*

Before we prove the safeness of the above reduction rule, we prove the following lemma.

▶ **Lemma 3.33.** *Let X be an optimal (clique, tree)-deletion set of G of size at most k and $A, \mathcal{B}, \widehat{\mathcal{B}}$ denote the vertex subsets obtained from Reduction Rule 3.32. Then, $v \in X$ or $A \subseteq X$.*

Proof. Let X be an optimal (clique, tree)-deletion set of G of size at most k. As the Lemma 2.4 (new q-expansion lemma) with $q = 4$ has been already applied in Reduction Rule 3.32 and the obtained sets are $A \subseteq (S \cup H_v) \setminus \{v\}$ and $\mathcal{B} \subseteq \mathcal{C}$ such that $N_{\mathcal{H}}(\mathcal{B}) \subseteq A$ (due to item (ii) of Lemma 2.4), any connected component $C \in \mathcal{B}$ can have neighbors only in $A \cup H_v \cup \{v\}$ in G. By Definition 3.30, every connected component $C \in \mathcal{C}$ has a vertex that is adjacent to v. If some component $C^* \in \mathcal{C}$ has two vertices adjacent to v, then there is a cycle that passes through v and the vertices of C^* but avoids H_v. This contradicts with Lemma 3.29 that H_v intersects all cycles passing through v. Hence, for every connected component $C \in \mathcal{C}$, there is exactly one vertex that is adjacent to v.

Suppose for the sake of contradiction that there is an optimal (clique, tree)-deletion set X^* of G of size at most k such that $v \notin X^*$ and $A \not\subseteq X^*$. Let $X^*_{\mathcal{B}}$ denotes the intersection of X^* with the vertices that are in the connected components in $\mathcal{B}$. We set

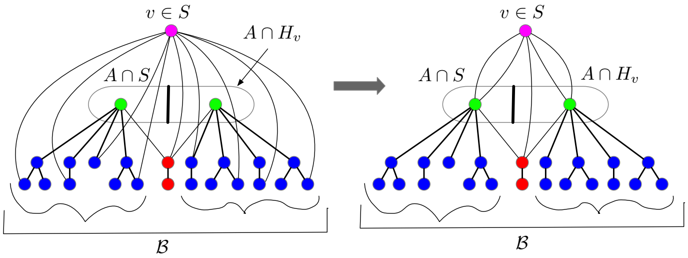


Figure 3 An illustration of Reduction Rule 3.32. The blue components are the components of $\mathcal{B}$ and the red component is the chosen component C for which the edge between u and C is not deleted. The blue components of $\mathcal{B}$ are the ones that are the endpoints of expansion $\widehat{M}$.

$\widehat{X} = (X^* \setminus X^*_{\mathcal{B}}) \cup (A \setminus X^*) \cup \{v\}$. We claim that $\widehat{X}$ is a (clique, tree)-deletion set of G and $|\widehat{X}| < |X^*|$. For the first part, note that the cycles hit by X^* but not $\widehat{X}$ must contain a vertex from $X^*_{\mathcal{B}}$. Such cycles must contain a vertex in $N_{\mathcal{H}}(\mathcal{B})$ as each component of $\mathcal{B}$ is a forest. It follows from the item (i) of Lemma 2.4 that $N_{\mathcal{H}}(\mathcal{B}) \subseteq A$ and $A \subseteq H_v \cup (S \setminus \{v\})$. Therefore, the neighbors of all vertices spanned by the connected components of $\mathcal{B}$ are contained in $A \cup \{v\}$. By construction of $\widehat{X}$, as $A \cup \{v\} \subseteq \widehat{X}$, it follows that $\widehat{X}$ is a (clique, tree)-deletion set of G.

Now, we claim that $|\widehat{X}| < |X^*|$. Since $A \not\subset X^*$ and $v \notin X^*$, there is $x \in A \setminus X^*$. Due to Lemma 2.4, there are four connected components C_1, C_2, C_3, C_4 in $\widehat{\mathcal{B}}$ such that v is adjacent to one vertex from each of C_1, C_2, C_3, C_4 and x is adjacent to some vertex in each of C_1, C_2, C_3, C_4. If v is adjacent to x, then $G[\{v, x\} \cup C_1 \cup C_2 \cup C_3 \cup C_4]$ has several cycles with a chord (or subdivision of diamonds) that contains $\{x, v\}$ and vertices from exactly two connected components from $\{C_1, C_2, C_3, C_4\}$. In particular for every pair $i, j \in \{1, 2, 3, 4\}$, there is a cycle with a chord containing v, x and vertices from C_i and C_j. Since $v, x \notin X^*$, it must be that X^* must have at least one vertex from at least three of these connected components $\{C_1, C_2, C_3, C_4\}$. As our updating procedure removes the vertices of $\mathcal{B}$ from X^* and adding vertices of $A \setminus X^*$, it follows that for every $x \in A \setminus X^*$, one vertex is added and at least three additional vertices appearing in the components of $\{C_1, C_2, C_3, C_4\}$ are removed from X^*. This ensures that $\widehat{X}$ has strictly lesser vertices than X^* contradicting the optimality of X^*. This completes the proof. ◀

We use the above lemma to prove that Reduction Rule 3.32 is safe.

▶ **Lemma 3.34.** *Reduction Rule 3.32 is safe.*

Proof. For the forward direction ($\Rightarrow$), let X be a (clique, tree)-deletion set of G with at most k vertices. We assume without loss of generality that X is an optimal (clique, tree)-deletion set of G. Due to Lemma 3.33, it follows that either $v \in X$ or $A \subseteq X$. For both the cases, observe that $G' - X$ is a subgraph of $G - X$. Hence, X is a (clique, tree)-deletion set of G' of size at most k'.

For the backward direction ($\Leftarrow$), let X' be a (clique, tree)-deletion set of G' of size at most $k'(= k)$. Note that in G', we have double edge between v and every vertex in A. Thus, $v \in X'$ or $A \subseteq X'$ to hit the cycles formed by these double edges. In case $v \in X'$, then the graphs $G - X'$ and $G' - X'$ are precisely the same. Therefore, X' is a (clique, tree)-deletion set of G as well. For the other case, we have that $A \subseteq X'$ but $v \notin X'$. Suppose for the sake of contradiction that some component of $G - X'$ is neither a clique, nor a tree. Then, $G - X'$ has an obstruction O. Since for any connected component $C' \in \mathcal{B}$, it must be that $N_{\mathcal{H}}(C') \subseteq A$, it follows that $N_G(C') \subseteq A \cup \{v\}$. By construction of G', an edge $uv \in E(G)$

is not an edge of G' if $u \in C'$ for some connected component $C' \in \widehat{\mathcal{B}}$. The obstruction O must contain an edge $uv \in E(G)$ such that $uv \notin E(G')$. Such an edge is possible only for some $C'' \in \widehat{\mathcal{B}}$. As v can have at most one neighbor in every component $C'' \in \mathcal{B}$ and the vertices of C'' are adjacent to only $A \cup \{v\}$ with $A \subseteq X'$, the only possible way uv edge can be part of an obstruction in $G - X'$ is a paw. Then, this obstruction O is a paw containing u and v. Since $|\mathcal{C}| > 4(|S| + |H_v|)$, and due to the condition (ii) of Proposition 2.4, $|\mathcal{C} \setminus \mathcal{B}| \leq 4|(S \cup H_v) \setminus A|$, it must be that $|\mathcal{B}| > 4|A|$. Therefore, there is at least one connected component $C \in \mathcal{B} \setminus \widehat{\mathcal{B}}$. Furthermore, Reduction Rule 3.32 has chosen not to delete the edge vu^* such that $u^* \in C$ and $vu^* \in E(G)$. But by construction of G', $vu^* \in E(G')$. Observe that $O^* = (O \setminus \{u\}) \cup \{u^*\})$ also induces a paw in the graph G. Then, u^* must be in the set X' as otherwise it would contradict that X' is a (clique, tree)-deletion set of X'. So, we set $X^* = (X' \setminus \{u^*\}) \cup \{v\}$ and clearly by construction $|X^*| = |X'|$. Since the neighrborhood of any connected component of $\mathcal{B}$ is contained in $A \cup \{v\}$, this ensures us that X^* is a (clique, tree)-deletion set of G. This completes the proof of the lemma. ◀

Observe that the above mentioned reduction rules do not increase the degree of v. When none of the above mentioned reduction rules are applicable, no connected component C (with at least three vertices) of $G[V_2]$ can have two leaves u and v both of which are pendant vertices in G. But, it is possible that C has one leaf u that is a pendant vertex of the whole graph G.

▶ **Lemma 3.35.** *Let G be the graph obtained after applying Reduction Rules 3.3-3.32 exhaustively. Then the number of vertices in V_2 is at most $1525k|S|$.*

Proof. We use N to denote the vertices of V_2 that are adjacent to some vertex of S. As Reduction Rule 3.32 is not applicable, for every $v \in S$, there are at most $60(k+1)(\leq 61k)$ edges incident to v with the other endpoint being in V_2 (hence in N). Hence, the number of vertices of N is at most $61k|S|$.

Let us bound the number of leaves in the forest $G[V_2]$ that is not in N. Since Reduction Rule 3.21 is exhaustively applied, such a leaf is adjacent to a vertex that is adjacent to some vertex in S (therefore, such vertices are in N). Since Reduction Rule 3.5 is exhaustively applied, two such leaves are not adjacent to the same vertex. Hence, we can define an injective function from the leaves of $G[V_2 \setminus N]$ to the internal vertices in the forest $G[V_2]$ that is in N.

Thus, the total number leaves in $G[V_2]$ is at most $2|N|$. Since, the number of vertices with degree at least 3 in $G[V_2]$ is at most the number of leaves in $G[V_2]$, the number of vertices of $G[V_2]$ with degree at least three is at most $2|N|$. Therefore, the sum of the number leaves in $G[V_2]$, and the number of vertices with degree at least 3 of $G[V_2]$ is at most $4|N|$. Additionally, there are some vertices of N that are neither counted as a leaf of $G[V_2]$ nor is counted as a vertex of degree at least three in $G[V_2]$. Number of such vertices is at most $|N|$.

It remains to bound the number of degree 2 vertices in the forest $G[V_2]$ that is not in N. Note that such vertices are also degree 2 vertices in G. Since Reduction Rules 3.7 and 3.10 are exhaustively applied, any degree-2-overbridge of $G[V_2]$ has size at most four. Each such degree-2-overbridge connects two vertices of $G[V_2]$ such that those two vertices are leaves, or vertices from N. We replace all such degree-2-overbridges by edges to get a forest H with at most $5|N|$ vertices, and thus edges. Since each edge of H corresponds to at most 4 vertices, we have at most $20|N|$ vertices of V_2 each of which have degree exactly two in G.

Thus the total number of vertices in $G[V_2]$ is bounded by $25|N|$. Since $|N| \leq 61k|S|$, the the total number of vertices in the forest $G[V_2]$ is bounded by $1525k|S|$. ◀

Combining Lemma 3.20 and Lemma 3.35, we are ready to prove our final result that we restate below.

▶ **Theorem 1.1.** Cliques or Trees Vertex Deletion *(CTVD) admits a kernel with* $\mathcal{O}(k^5)$ *vertices.*

Proof. Given the input instance (G, k), the kernelization algorithm invokes Reduction Rules 3.3-3.32 exhaustively. Let (G', k') denotes the output instance such that S' is a (clique, tree)-deletion set of G' with at most $4k$ vertices. Suppose that $V_1 \subseteq G' - S'$ denotes the vertices such that every connected component of $G[V_1]$ is a clique and $V_2 \subseteq G' - S'$ denotes the vertices such that every connected component of $G[V_2]$ is a tree. Since Reduction Rules 3.3-3.18 are not applicable, it follows from Lemma 3.20 that $|V_1|$ is $\mathcal{O}(k^5)$. Additionally, as Reduction Rules 3.3-3.32 are not applicable, it follows from Lemma 3.35 that $|V_2| \leq 1525k|S'| = 6100k^2$. Therefore, the total number of vertices in G' is $|S'| + |V_1| + |V_2|$ which is $\mathcal{O}(k^5)$. ◀

4 Conclusions and Future Research

Our paper initiates a study of polynomial kernelization for vertex deletion to pairs of scattered graph classes. One natural open question is to improve the size of our kernel, e.g. to $\mathcal{O}(k^3)$ vertices. We believe that such a result is possible to achieve, but we suspect that it would require new techniques to develop such results. Jacob et al. [26] have provided an $\mathcal{O}^*(4^k)$-time algorithm for Cliques or Trees Vertex Deletion. It would also be interesting to design an FPT algorithm where the base of the exponent is (substantially) improved from 4. On a broader level, it would be interesting to explore the possibility of getting a polynomial kernel for problems where the objective is to delete a set of at most k vertices so that the connected components would belong to other interesting pairs of graph classes, such as (interval graph, trees), and (chordal graph, bipartite permutation). In addition, vertex/edge deletion to scattered graph classes are also interesting from approximation algorithms perspective. In fact, it would be interesting to improve the approximation guarantee of Proposition 3.1 that is also an open problem. Additionally, the dual version of this problem, i.e. “packing vertex-disjoint obstructions to the scattered class of cliques and trees” is also interesting from the perspective of parameterized complexity. The same problem can be considered as packing vertex-disjoint induced subgraphs that are paws or diamonds or cycles of length at least 4. A natural approach to solve this problem requires to design an Erdos-Posa style theorem for packing obstructions for scattered class of cliques and trees. Finally, a more general open problem is to identify pairs of graph classes (Π_1, Π_2) for which vertex deletion to Π_1 as well as vertex deletion to Π_2 admits polynomial sized kernels, but (Π_1, Π_2)-Deletion does not admit a polynomial kernel.

References

1 Jasine Babu, R. Krithika, and Deepak Rajendraprasad. Packing arc-disjoint 4-cycles in oriented graphs. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.FSTTCS.2022.5.

2 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for P_5-free graphs. *Algorithmica*, 81(4):1342–1369, 2019. doi:10.1007/S00453-018-0474-X.

3 Anudhyan Boral, Marek Cygan, Tomasz Kociumaka, and Marcin Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 58(2):357–376, 2016. doi:10.1007/S00224-015-9631-7.

4 Hajo Broersma, Jirí Fiala, Petr A. Golovach, Tomás Kaiser, Daniël Paulusma, and Andrzej Proskurowski. Linear-time algorithms for scattering number and hamilton-connectivity of interval graphs. *J. Graph Theory*, 79(4):282–299, 2015. doi:10.1002/JGT.21832.

5 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

6 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015. doi:10.1145/2629595.

7 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/S00453-015-0014-X.

8 Jianer Chen. Vertex cover kernelization. In *Encyclopedia of Algorithms*, pages 2327–2330. Springer, 2016. doi:10.1007/978-1-4939-2864-4_460.

9 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/J.TCS.2010.06.026.

10 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of mathematics*, pages 51–229, 2006.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

12 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Jan Derbisz, Lawqueen Kanesh, Jayakrishnan Madathil, Abhishek Sahu, Saket Saurabh, and Shaily Verma. A polynomial kernel for bipartite permutation vertex deletion. *Algorithmica*, 84(11):3246–3275, 2022. doi:10.1007/S00453-022-01040-9.

15 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

17 Maël Dumas and Anthony Perez. An improved kernelization algorithm for trivially perfect editing. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.15.

18 Bruno Escoffier, Laurent Gourvès, and Jérôme Monnot. Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs. *J. Discrete Algorithms*, 8(1):36–49, 2010. doi:10.1016/J.JDA.2009.01.005.

19 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. doi:10.1145/3293466.

20 Fedor V Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM Journal on Discrete Mathematics*, 27(4):1964–1976, 2013. doi:10.1137/12089051X.

21 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Trans. Algorithms*, 13(2):29:1–29:32, 2017. doi:10.1145/3014587.

22 Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in claw-free graphs. *SIAM J. Discret. Math.*, 29(1):348–375, 2015. doi:10.1137/140963200.

23 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

24 Pinar Heggernes, Dieter Kratsch, and Daniel Meister. Bandwidth of bipartite permutation graphs in polynomial time. *J. Discrete Algorithms*, 7(4):533–544, 2009. doi:10.1016/J.JDA.2008.11.001.

25 Ashwin Jacob, Jari J. H. de Kroon, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes I - case of finite number of graph classes. *J. Comput. Syst. Sci.*, 138:103460, 2023. doi:10.1016/J.JCSS.2023.05.005.

26 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes II - improved FPT algorithms for deletion to pairs of graph classes. *J. Comput. Syst. Sci.*, 136:280–301, 2023. doi:10.1016/J.JCSS.2023.03.004.

27 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Expansion lemma - variations and applications to polynomial-time preprocessing. *Algorithms*, 16(3):144, 2023. doi:10.3390/A16030144.

28 Hugo Jacob, Thomas Bellitto, Oscar Defrain, and Marcin Pilipczuk. Close relatives (of feedback vertex set), revisited. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.21.

29 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Wlodarczyk. Single-exponential FPT algorithms for enumerating secluded f-free subgraphs and deleting to scattered graph classes. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.42.

30 Matthew Johnson, Giacomo Paesani, and Daniël Paulusma. Connected vertex cover for $(sP_1 + P_5)$-free graphs. *Algorithmica*, 82(1):20–40, 2020. doi:10.1007/S00453-019-00601-9.

31 Tereza Klimosová, Josef Malík, Tomás Masarík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $(P_r + P_s)$-free graphs. *Algorithmica*, 82(7):1833–1858, 2020. doi:10.1007/S00453-020-00675-W.

32 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014. doi:10.1016/J.IPL.2014.05.001.

33 Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set on at-free graphs. *Discret. Appl. Math.*, 156(10):1936–1947, 2008. doi:10.1016/J.DAM.2007.10.006.

34 Cornelis Lekkeikerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

35 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

36 Barnaby Martin, Daniël Paulusma, and Erik Jan van Leeuwen. Disconnected cuts in claw-free graphs. *J. Comput. Syst. Sci.*, 113:60–75, 2020. doi:10.1016/J.JCSS.2020.04.005.

37 George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

38 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Trans. Algorithms*, 2(3):403–415, 2006. doi:10.1145/1159892.1159898.

39 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

Hardness Amplification for Dynamic Binary Search Trees

Shunhua Jiang
Columbia University, New York, NY, USA

Victor Lecomte
Stanford University, CA, USA

Omri Weinstein
The Hebrew University of Jerusalem, Israel

Sorrachai Yingchareonthawornchai
The Hebrew University of Jerusalem, Israel

Abstract

We prove direct-sum theorems for Wilber's two lower bounds [Wilber, FOCS'86] on the cost of access sequences in the binary search tree (BST) model. These bounds are central to the question of dynamic optimality [Sleator and Tarjan, JACM'85]: the *Alternation* bound is the only bound to have yielded online BST algorithms beating $\log n$ competitive ratio, while the *Funnel* bound has repeatedly been conjectured to exactly characterize the cost of executing an access sequence using the optimal tree [Wilber, FOCS'86, Kozma'16], and has been explicitly linked to splay trees [Levy and Tarjan, SODA'19]. Previously, the direct-sum theorem for the Alternation bound was known only when approximation was allowed [Chalermsook, Chuzhoy and Saranurak, APPROX'20, ToC'24].

We use these direct-sum theorems to amplify the sequences from [Lecomte and Weinstein, ESA'20] that separate between Wilber's Alternation and Funnel bounds, increasing the Alternation and Funnel bounds while optimally maintaining the separation. As a corollary, we show that Tango trees [Demaine et al., FOCS'04] are optimal among any BST algorithms that charge their costs to the Alternation bound. This is true for *any* value of the Alternation bound, even values for which Tango trees achieve a competitive ratio of $o(\log \log n)$ instead of the default $O(\log \log n)$. Previously, the optimality of Tango trees was shown only for a limited range of Alternation bound [Lecomte and Weinstein, ESA'20].

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Online algorithms

Keywords and phrases Data Structures, Amortized Analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.42

Related Version *Full Version*: https://arxiv.org/pdf/2411.14387

Funding This research was supported by the European Research Council (ERC) Starting grant CODY 101039914.

Acknowledgements This work was done in part while the authors (SJ, OW, SY) were visiting the Simons Institute for the Theory of Computing, UC Berkeley.

1 Introduction

Direct Sum theorems assert a lower bound on a certain complexity measure $\mathcal{C}$ of a *composed*[1] problem $f \circ g$ in terms of the individual complexities of f and g, ideally of the form $\mathcal{C}(f \circ g) \approx \mathcal{C}(f) + \mathcal{C}(g)$. Direct Sums have a long history in complexity theory, as they provide

[1] Formally speaking, direct-sum problems pertain to the complexity of solving k separate copies of a problem f, rather than computing a composed function of k copies $g(f(x_1), \ldots, f(x_k))$, but it is common to refer to both variations of the k-fold problem as direct-sums [23].

a *black-box* technique for amplifying the hardness of computational problems $\mathcal{C}(f^{\circ k}) \gtrsim k \cdot \mathcal{C}(f)$, and are the most promising approach for proving several holy-grail lower bounds in complexity theory [23, 20, 35, 2, 24]. Moreover, a "tensorization" property of $\mathcal{C}$ under composition allows to "lift" the problem and leverage its asymptotic behavior (e.g., concentration), which is not present in the single-copy problem – this feature has been demonstrated and exploited in various models, including combinatorial Discrepancy [28, 38], Richness of data structure problems [33], decision trees [35] and rank [24] to mention a few. Despite their powerful implications, (strong) direct-sum scaling of composed problems are often simply false [34, 39, 37], and highly depend on the underlying computational model.

In this paper, we study direct sums in the *online BST model*, motivated by the *dynamic optimality* conjecture of Sleator and Tarjan [40]. The dynamic optimality conjecture postulates the existence of an *instance optimal* binary search tree algorithm (BST), namely, an online self-adjusting BST whose running time[2] matches the best possible running time *in hindsight* for any sufficiently long sequence of queries. More formally, denoting by $\mathcal{T}(X)$ the operational time of a BST algorithm $\mathcal{T}$ on an access sequence $X = (x_1, \ldots, x_m) \in [n]^m$ of keys to be searched, the conjecture says that there is an online BST $\mathcal{T}$ such that $\forall X$, $\mathcal{T}(X) \leq O(\mathsf{OPT}(X))$, where $\mathsf{OPT}(X) := \min_{\mathcal{T}'} \mathcal{T}'(X)$ denotes the optimal offline cost for X. In their seminal paper, Sleator and Tarjan [40] conjectured that *splay trees* are $O(1)$-competitive; A more recent competitor, the *GreedyFuture* algorithm [30, 15, 31], also forms a compelling candidate for constant-competitive dynamic optimality. However, the near-optimality of both Splay trees and GreedyFuture was proven only in special cases [41, 18, 32, 7, 8, 21, 10, 13], and they are not known to be $o(\log n)$-competitive for general access sequences X (note that every balanced BST is trivially $O(\log n)$-competitive). After 35 years of active research, the best provable bound to date is an $O(\log \log n)$-competitive BST, starting with *Tango trees* [16], among other $O(\log \log n)$-competitive BST algorithms [5, 42, 6]. Interestingly, this progress was made possible due the development of *lower bounds* in the BST model, as we discuss next.

Indeed, a remarkable feature of the BST model – absent from general computational models (e.g., word-RAM) – is that it allows for nontrivial lower bounds on the search time of a *fixed* query sequence X: In general models, lower bounds against a specific input X do not make much sense as the best algorithm in hindsight can simply "store and read-off the answer" for X. Nevertheless, in the BST model, even an all-knowing binary search tree must pay the cost of traversing the root-to-leaf path to retrieve keys. For example, there are classical examples of deterministic access sequences (e.g., *bit-reversal* sequence [43]) that require the worst case $\Omega(m \log n)$ total search time. This feature is what makes instance-optimality in the BST model an intriguing possibility. Our work focuses on two classic lower bounds due to Wilber [43], the Alternation and Funnel bounds (a.k.a, Wilber's first and second bounds), which are central to the aforementioned developments.

The Alternation and Funnel Bounds. Essentially all BST lower bounds are derived from a natural geometric interpretation of the access sequence $X = (X_1, \ldots, X_m)$ as a point set on the plane, mapping the i^{th} access X_i to point (X_i, i) ([15, 22], see Figure 1). The earliest lower bounds on $\mathsf{OPT}(X)$ were proposed in an influential paper of Wilber [43]. The *alternation bound* $\mathsf{Alt}_{\mathcal{T}}(X)$ counts the total number of left/right alternations obtained by searching the keys $X = (X_1, \ldots, X_m)$ on a *fixed* (static) binary search tree $\mathcal{T}$, where alternations are summed up over all nodes $v \in \mathcal{T}$ of the "reference tree" $\mathcal{T}$ (see Figure 3

[2] i.e. the number of pointer movements and tree rotations performed by the BST

and the formal definition in Section 2). Thus, the Alternation bound is actually a family of lower bounds, optimized by the choice of the reference tree $\mathcal{T}$, and we henceforth define $\mathsf{Alt}(X) := \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(X)$. The Alternation bound plays a key role in the design and analysis of Tango trees and their variants [16, 42]. In fact, *all* non-trivial $o(\log n)$-competitive BST algorithms [16, 5, 42, 6, 11] rely on the Alternation bound.

In the same paper, Wilber proposed another lower bound – the *Funnel Bound* $\mathsf{Funnel}(X)$ – which is less intuitive and can be defined by the following process: Consider the geometric view $\{(X_i, i)\}_{i\in[m]}$ of the simple "move-to-root" algorithm that simply rotates each searched key X_i to the root by a series of single rotations. Then $\mathsf{Funnel}(X_i, i)$ is exactly the number of *turns* on the path from the root to X_i right before it is accessed [1, 22]. The Funnel bound is then defined as $\mathsf{Funnel}(X) := \sum_{i=1}^{m} \mathsf{Funnel}(X_i, i)$. This view emphasizes the *amortized* nature of the Funnel bound: at any point, there could be linearly many keys in the tree that are only *one* turn away from the root, so one can only hope to achieve this bound in some amortized fashion.

The Funnel bound has been repeatedly conjectured to tightly characterize the cost of an offline optimal algorithm [43, 25, 6, 27]. Recently, Lecomte and Weinstein [27] proved that the funnel bound is *rotation-invariant*, meaning that the bound is preserved when the geometric representation of the input sequence is rotated by 90 degrees. This property also holds for an optimal algorithm [15], giving another evidence that the Funnel bound might give a tight characterization of the cost of an offline optimal algorithm. While the Funnel bound does not have an algorithmic interpretation like $\mathsf{Alt}(X)$, Levy and Tarjan [29] recently observed interesting similarities between Splay trees and the Funnel bound. The core difficulty in converting $\mathsf{Funnel}(X)$ into a BST algorithm is its highly *amortized* nature (also a feature of Splay trees), compared to the Alternation bound which gives a point-wise lower bound on the retrieval time of $X_i \in X$. As such, understanding the mathematical properties of the Funnel bound is important in its own right.

Access Sequence Composition and Known Direct-Sum Results. Informally, Direct Sum theorems assert a lower bound on the complexity measure of solving R copies of a problem in a given computational model, in terms of the cost of solving a single copy, ideally $C(f^{\circ R}) \gtrsim \Omega(R) \cdot C(f)$, where $f^{\circ R}$ denotes certain R-copy *composed* problem. Indeed, the precise notion of composition we use here (a-la [9]) is crucial, as direct-sum theorems are subtle and often turn out to be *false* [34].

A natural definition of sequence-composition in the BST model was introduced by Chalermsook et al. [9]. Let $X^{(1)}, \ldots, X^{(\ell)}$ be a sequence of ℓ access sequences where $X^{(i)} \in [n_i]^{m_i}$. That is, each sequence $X^{(i)}$ has n_i keys and m_i accesses where $m := \sum_i m_i, n := \sum_i n_i$. We view each sequence $X^{(i)}$ as the i^{th} queue where we dequeue elements by the order of the sequence $X^{(i)}$ (i.e., in FIFO order). Let $\widetilde{X} \in [\ell]^m$ be a sequence with keys in $[\ell]$ such that every $j \in [\ell]$ appears exactly m_j times. We can view $\widetilde{X}$ as a *template* which defines the ordering of dequeue operations among the ℓ queues. We define the *composed sequence* $X := \widetilde{X}(X^{(1)}, \ldots, X^{(\ell)}) \in [n]^m$ as follows. For each $t = 1$ to m, $X_t := q_t + \sum_{i<\tilde{X}_t} n_i$ where q_t is the next element dequeued from $X^{(\tilde{X}_t)}$. We refer the reader to Section 2 for the precise definition.

The direct sum results for the optimal cost are well understood with applications to proving non-trivial bounds of binary search trees. In [9], they prove (approximate) *subadditivity* of the optimal cost on composed sequences. That is,

$$\mathsf{OPT}(X) \le 3 \cdot \mathsf{OPT}\left(\widetilde{X}\right) + \sum_j \mathsf{OPT}\left(X^{(j)}\right).$$

The subadditivity of optimal cost finds application in proving the linear optimal bounds for "grid" sequences, and a strong separation in the hierarchy of *lazy finger bounds* [9]. On the other hand, [6] recently proved *superadditivity* of the optimal cost on composed sequences. That is,

$$\mathsf{OPT}(X) \geq \mathsf{OPT}\Big(\widetilde{X}\Big) + \sum_j \mathsf{OPT}\Big(X^{(j)}\Big). \tag{1}$$

The superadditivity of optimal cost finds an application in designing a new $O(\log\log n)$-competitive online BST algorithm based on purely geometric formulation [6].

However, the direct sum results for Wilber's bounds are poorly understood despite their importance to the pursuit of dynamic optimality. The only published work that we are aware of is the approximate subadditivity of the Alternation bound [6]. That is, they proved that

$$\mathsf{Alt}(X) \leq 4 \cdot \mathsf{Alt}\Big(\widetilde{X}\Big) + 8 \cdot \sum_j \mathsf{Alt}\Big(X^{(j)}\Big) + O(|X|). \tag{2}$$

Their proof is quite involved, and it is based on geometric arguments and the probabilistic method. This finds applications in proving the separation between the Alternation and Funnel bounds. In [6], they used approximate subadditivity of the Alternation bound (Equation (2)) to prove a near-optimal separation between the Alternation and Funnel bounds. That is, they constructed a sequence Y such that the gap between $\mathsf{Alt}(Y)$ and $\mathsf{Funnel}(Y)$ is as large as $\Omega(\frac{\log\log n}{\log\log\log n})$. This gap is nearly optimal because the upper bound of Tango tree [16] implies that the gap must be $O(\log\log n)$. This gap has been closed by an independent work by Lecomte and Weinstein [27], proving the optimal separation between the Alternation and Funnel bounds. That is, they constructed an instance Y such that the gap between $\mathsf{Alt}(Y)$ and $\mathsf{Funnel}(Y)$ is as large as $\Omega(\log\log n)$.

Furthermore, [6] also used the approximate subadditivity of the Alternation bound (Equation (2)) to prove the $\Omega(\frac{\log\log n}{\log\log\log n})$ gap between $\mathsf{Alt}(Y)$ and $\mathsf{cGB}(Y)$ where $\mathsf{cGB}(Y)$ denotes the *Consistent Guillotine Bound* (cGB), a lower bound measure that is an extension of $\mathsf{Alt}(Y)$.

1.1 Our Results

We prove that the Alternation bound is subadditive whereas the Funnel bound is superadditive for composed sequences. More precisely, we prove the following theorem.

▶ **Theorem 1** (Direct-Sum Theorem for Wilber's Bounds). *Let* $X := \widetilde{X}(X^{(1)}, \dots, X^{(l)})$ *be a composed sequence. Then*

- $\mathsf{Alt}(X) \leq \mathsf{Alt}\Big(\widetilde{X}\Big) + \sum_j \mathsf{Alt}\big(X^{(j)}\big) + O(|X|)$, *and*
- $\mathsf{Funnel}(X) \geq \mathsf{Funnel}\Big(\widetilde{X}\Big) + \sum_j \mathsf{Funnel}\big(X^{(j)}\big) - O(|X|)$.

Our proof of the subadditivity of the Alternation bound is simpler and yields stronger bounds than the proof in [6] (Theorem 3.6 in their arXiv version). Direct sum theorems are a natural black-box tool for hardness amplification, as they effectively reduce complex lower bounds to a simpler "one-dimensional" problem. Indeed, as a showcase application, we use the base-case separation proved in [27] along with Theorem 1 to amplify both Wilber's bounds. Let $\overline{\mathsf{Alt}}(X) := \mathsf{Alt}(X)/|X|$, and $\overline{\mathsf{Funnel}}(X) := \mathsf{Funnel}(X)/|X|$. They proved that there is a sequence Y such that

$$\overline{\mathsf{Alt}}(Y) \leq O(1), \text{but}$$
$$\overline{\mathsf{Funnel}}(Y) \geq \Omega(\log\log n).$$

Note that the sequence Y is *easy* w.r.t. the Alternation bound since $\overline{\mathsf{Alt}}(Y) \leq O(1)$. We use the sequence Y as a base-case and apply Theorem 1 to construct *hard* sequence w.r.t. the funnel bound while maintaining the separation as in the following theorem [3].

▶ **Theorem 2** (Hardness Amplification). *There is a constant $K > 0$ such that for any n of the form 2^{2^r} and any power-of-two $R \leq \frac{\log n}{K}$, there is a sequence $Y_n^{\circ R} \in [n]^{m'}$ with $m' \leq \mathrm{poly}(n)$ such that*

$$\overline{\mathsf{Alt}}(Y_n^{\circ R}) \leq O(R)$$
$$\overline{\mathsf{Funnel}}(Y_n^{\circ R}) \geq \Omega\left(R \log\left(\frac{\log n}{R}\right)\right).$$

Remark 1. We emphasize that the approximate subadditivity of the Alternation bound (Equation (2)) is not sufficient for such hardness amplification. On the other hand, one could also use the superadditivity of the optimal cost (Equation (1)) instead of the Funnel bound to prove hardness amplification.

Tightness of the Separation. As a corollary of Theorem 2, we can derive the following trade-offs between the multiplicative and additive factors for the Alternation bound.

▶ **Theorem 3.** *Let $\alpha, \beta : \mathbb{N} \to \mathbb{R}_{\geq 1}$ be any functions such that some BST algorithm achieves an amortized cost of $\alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n)$ for all access sequences X over n keys. Then $\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$.*

As we discuss below, the trade-offs are tight with the matching upper bounds from Tango Trees (which can be derived directly from Tango trees). For convenience, we present self-contained BST algorithms with the matching upper bounds .

▶ **Lemma 4.** *There is a BST algorithm that takes an integer $k \leq \log n$ as a parameter and serve the sequence $X = (X_1, \cdots, X_m)$ with the total access time of*

$$O\left(\left(\mathsf{Alt}(X) + m \cdot \frac{\log n}{k}\right) \cdot (\log k + 1)\right).$$

For the reason of space, we defer the proof of Lemma 4 to the full version.

In this algorithm, the additive cost is $\Theta\left(\frac{\log n \log k}{k}\right)$ and multiplicative cost $\Theta(\log k)$. By Theorem 3, if $\beta(n) = \Theta\left(\frac{\log n \log k}{k}\right)$, then we have

$$\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right) = \Omega\left(\log\left(\frac{k}{\log k}\right)\right) = \Omega(\log k),$$

so our trade-off is optimal for any sufficiently large $k \leq \log n$.

Optimality of Tango Trees. As another corollary of Theorem 2, we prove the optimality of Tango Trees among any algorithm charging its cost to Wilber's Alternation bound for all values of the $\overline{\mathsf{Alt}}(X)$. Note that $\overline{\mathsf{Alt}}(X) \leq O(\log n)$. Previously, the optimality of Tango trees is known only when $\overline{\mathsf{Alt}}(X) = O(1)$ [27].

[3] This can be viewed as hardness amplification because the new sequence becomes harder from the optimum's point of view without losing the gap too much.

The basic idea of Tango Trees is to "mimic" Wilber's alternation bound via a BST, by dynamically maintaining a partition of the reference tree $\mathcal{T}$ into *disjoint paths*, formed by designating, for each node $x \in \mathcal{T}$, the unique "preferred" descendant in $\mathcal{T}$ (left or right) which was *acessed most recently.* Since each "preferred path" has length $|p| \leq \text{depth}(\mathcal{T}) \leq \log n$, every *path* can itself be stored in a BST, so assuming these paths can be dynamically maintained (under split and joins), searching for the predecessor of a key x_i *inside* each path only takes $O(\log\log n)$ time, until the search "falls-off" the current preferred path and switches to a different one. The key observation is that this "switch" can be charged to $\mathsf{Alt}_{\mathcal{T}}(X)$, as it certifies a new alternation in Wilber's lower bound, hence OPT must pay for this move as well. This elegant argument directly leads to the aforementioned $O(\log\log n) \cdot \mathsf{OPT}(X)$ search time.

The analysis of Tango trees relies on charging the algorithm's cost to the Alternation bound. One may ask if the bound can be improved using a clever algorithm (not necessarily Tango trees) so that we can charge $o(\log\log n)$ factor to the Alternation bound. Unfortunately, the answer is no as there are known examples of access sequences $\tilde{X}$ with $\mathsf{Alt}(\tilde{X}) = O(m)$ but $\mathsf{OPT}(\tilde{X}) = \Theta(m \log\log n)$ [27, 6]. In light of this, Tango trees are indeed off by a factor $\Theta(\log\log n)$ from $\mathsf{Alt}(\tilde{X})$. Interestingly, when $\mathsf{OPT}(X) \gtrsim m\frac{\log n}{2^{o(\log\log n)}}$, one can do somewhat better. Let $\overline{\mathsf{OPT}}(X) := \mathsf{OPT}(X)/|X|$. The Tango tree, presented by [16] (see the discussion in their Section 1.5), has a competitive ratio of

$$O\left(1 + \log \frac{\log n}{\overline{\mathsf{OPT}}(X)}\right) = o(\log\log n). \tag{3}$$

The condition that allows $o(\log\log n)$-competitiveness is rather narrow: the amortized optimal cost $\overline{\mathsf{OPT}}(X)$ must be very close to $\log n$ to achieve $o(\log\log n)$-competitiveness. Can we achieve $o(\log\log n)$-competitiveness with a wider range of $\overline{\mathsf{OPT}}(X)$ using $\mathsf{Alt}(X)$?

Unfortunately, the answer is no. More generally, we prove a matching lower bound: the competitive ratio of any BST algorithm based on the Alternation bound must be at least $\Omega(\log \frac{\log n}{\overline{\mathsf{OPT}}(X)})$, matching to the upper bound of the Tango tree by Equation (3). More precisely, we prove the following theorem whose proof is a small modification of the proof in Theorem 3.

▶ **Theorem 5.** *Let $\alpha : \mathbb{N} \times \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ be any function such that some BST algorithm achieves an amortized cost of $\alpha\big(n, \overline{\mathsf{OPT}}(X)\big) \cdot \big(\overline{\mathsf{Alt}}(X) + 1\big)$ for all access sequences X over n keys. Then $\alpha(n, s) \geq \Omega\Big(\log\Big(\frac{\log n}{s}\Big)\Big)$.*

As a corollary of Theorem 5,[4] the lower bound of $\Omega(\log \frac{\log n}{\overline{\mathsf{OPT}}(X)})$ follows by setting s to be within constant factor of $\overline{\mathsf{OPT}}(X)$. This holds for every BST algorithm based on the Alternation bound. With the matching upper bound by Equation (3), Tango tree optimally utilizes the Alternation bound.

1.2 Further Related Work

Splay trees and GreedyFuture are prime candidates for the dynamic optimality conjecture since they both satisfy many important properties of dynamic trees including static optimality [40], working-set property [40, 19], dynamic finger property [14, 12], and more (see the

[4] We remark that we cannot set $\beta(n) = \overline{\mathsf{OPT}}(X)$ in Theorem 3 because $\beta(n)$ does not depend on X and, even if $\overline{\mathsf{OPT}}(X)$ is a function of n, the construction of the sequence X depends on $\beta(n)$. We need the lower bound of Theorem 5.

surveys [22, 25] for an overview of the results in the field). Although they are not yet known to have $o(\log n)$-competitiveness, they have substantially better bounds for special cases. For example, both Splay trees and GreedyFuture are dynamically optimal for *sequential access* sequences [18, 19]. For *deque sequences*, Splay trees are $O(\alpha^*(n))$-competitive [32] whereas GreedyFuture is $O(\alpha(n))$-competitive [10]. Here, $\alpha(n)$ denotes the inverse Ackermann function and $\alpha^*(n)$ is the iterated function of $\alpha(n)$. The sequential and deque sequences are special cases of the *pattern-avoiding access* sequences [8]. For any fixed-size pattern, GreedyFuture is $O(2^{(1+o(1))\alpha(n)})$-competitive for pattern-avoiding access sequences [13]. It was shown recently that an optimal BST algorithm takes $O(n)$ total cost for any fixed pattern [3]. The bounds for specific classes of patterns can be improved if preprocessing is allowed [8, 21] (i.e., the initial tree can be set before executing the online search sequences). Recently, a slight modification of GreedyFuture was shown to be $O(\sqrt{\log n})$-competitive [11]. An important application of GreedyFuture (or any competitive online BSTs) includes adaptive sorting using treesort [4, 13] and heapsort [26].

The lower bounds in the literature other than Wilber's bounds include the *maximum independent rectangle* and *SignedGreedy* bounds [15], which subsume Wilber's Alternation and Funnel bounds. A similar lower bound was presented by Derryberry et al. [17]. Recently, *Guillotine* Bound [6] was introduced, which is a generalization of Wilber's Alternation bound. Unfortunately, it is unclear how to design an algorithm that utilizes these bounds. Recently, Sadeh and Kaplan [36] proved that the competitive ratio of GreedyFuture cannot be less than 2 for the multiplicative factor, or $o(m \log\log n)$ for the additive factor.

1.3 Paper Organization

We first describe terminologies and notations in Section 2. We prove the direct-sum results for Wilber's bounds (Theorem 1) in Section 3. In Section 4, we prove the hardness amplification of Wilber's bounds while maintaining their separation (Theorem 2) and we also prove Theorem 3.

2 Preliminaries

We follow notations and terminologies from [27].

▶ **Definition 6** (access sequence)**.** *An access sequence is a finite sequence $X = (X_1, \dots, X_m) \in S^m$ of values from a finite set of keys $S \subseteq \mathbb{R}$. Usually, we let $S := [n]$.*

To make our definitions and proofs easier, we will work directly in the geometric representation of access sequences as (finite) sets of points in the plane $\mathbb{R}^2$.

▶ **Definition 7** (geometric view)**.** *Any access sequence $X = (X_1, \dots, X_m) \in S^m$ can be represented as the set of points $G_X := \{(X_t, t) \mid t \in [m]\}$, where the x-axis represents the key and the y-axis represents time (see Figure 1).*

By construction, in G_X, no two points share the same y-coordinate. We will say such a set has "distinct y-coordinates". In addition, we note that it is fine to restrict our attention to sequences X without repeated values.[5] The geometric view G_X of such sequences also has no two points with the same x-coordinate. We will say that such a set has "distinct x- and y-coordinates".

[5] Indeed, Appendix E in [8] gives a simple operation that transforms any sequence X into a sequence $\mathrm{split}(X)$ without repeats such that $\mathsf{OPT}(\mathrm{split}(X)) = \Theta(\mathsf{OPT}(X))$. Thus if we found a tight lower bound $L(X)$ for sequences without repeats, a tight lower bound for general X could be obtained as $L(\mathrm{split}(X))$.

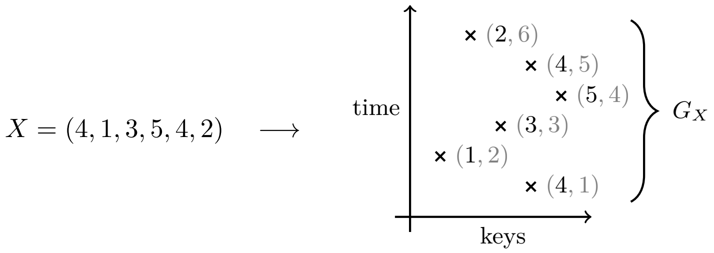


Figure 1 transforming X into its geometric view G_X.

▶ **Definition 8** (x- and y-coordinates)**.** *For a point $p \in \mathbb{R}^2$, we will denote its x- and y-coordinates as $p.x$ and $p.y$. Similarly, we define $P.x \coloneqq \{p.x \mid p \in P\}$ and $P.y \coloneqq \{p.y \mid p \in P\}$.*

We start by defining the *mixing value* of two sets: a notion of how many two sets of numbers are interleaved. It will be useful in defining both the alternation bound and the funnel bound. We define it in a few steps.

▶ **Definition 9** (mixing string)**.** *Given two disjoint finite sets of real numbers L, R, let $\text{mix}(L, R)$ be the string in $\{\texttt{L}, \texttt{R}\}^*$ that is obtained by taking the union $L \cup R$ in increasing order and replacing each element from L by $\texttt{L}$ and each element from R by $\texttt{R}$. For example, $\text{mix}(\{2,3,8\},\{1,5\}) = \texttt{RLLRL}$.*

▶ **Definition 10** (number of switches)**.** *Given a string $s \in \{\texttt{L}, \texttt{R}\}^*$, we define $\#\text{switches}(s)$ as the number of side switches in s. Formally,*

$$\#\text{switches}(s) \coloneqq \#\{t \mid s_t \neq s_{t+1}\}.$$

For example, $\#\text{switches}(\texttt{LLLRLL}) = 2$. Note that if we insert characters into s, $\#\text{switches}(s)$ can only increase.

▶ **Definition 11** (mixing value)**.** *Let $\text{mixValue}(L, R) \coloneqq \#\text{switches}(\text{mix}(L, R))$ (see Figure 2).*

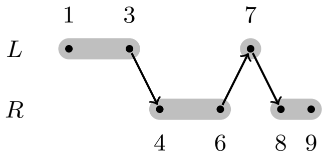


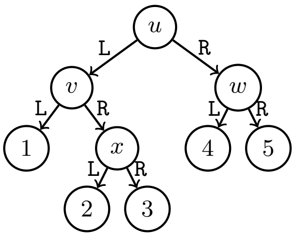
Figure 2 a visualization of $\text{mixValue}(\{1,3,7\},\{4,6,8,9\}) = 3$.

The mixing value has some convenient properties, which we will use later:

▶ **Fact 12** (properties of mixValue)**.** *Function $\text{mixValue}(L, R)$ is:*
(a) *symmetric: $\text{mixValue}(L, R) = \text{mixValue}(R, L)$;*
(b) *monotone: if $L_1 \subseteq L_2$ and $R_1 \subseteq R_2$, then $\text{mixValue}(L_1, R_1) \leq \text{mixValue}(L_2, R_2)$;*
(c) *superadditive under concatenation: if $L_1, R_1 \subseteq (-\infty, x]$ and $L_2, R_2 \subseteq [x, +\infty)$, then $\text{mixValue}(L_1 \cup L_2, R_1 \cup R_2) \geq \text{mixValue}(L_1, R_1) + \text{mixValue}(L_2, R_2)$.*
Finally, $\text{mixValue}(L, R) \leq 2 \cdot \min(|L|, |R|) + 1$.

The definitions of Wilber's Alternation and Funnel bounds $(\mathsf{Alt}(X), \mathsf{Funnel}(X))$ are standard in the literature.

We now give precise definitions of Wilber's two bounds.[6]



reference tree $\mathcal{T}$

Node	Link used by each access	Group by letter	#
u	$\mathtt{R}, \mathtt{L}, \mathtt{L}, \mathtt{R}, \mathtt{R}, \mathtt{L}$	$[\mathtt{R}], [\mathtt{L}, \mathtt{L}], [\mathtt{R}, \mathtt{R}], [\mathtt{L}]$	3
v	$\mathtt{L}, \mathtt{R}, \mathtt{R}$	$[\mathtt{L}], [\mathtt{R}, \mathtt{R}]$	1
w	$\mathtt{L}, \mathtt{R}, \mathtt{L}$	$[\mathtt{L}], [\mathtt{R}], [\mathtt{L}]$	2
x	$\mathtt{R}, \mathtt{L}$	$[\mathtt{R}], [\mathtt{L}]$	1
Total			7

Figure 3 For access sequence $X = (4, 1, 3, 5, 4, 2)$ and reference tree $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(X) = 7$.

▶ **Definition 13** (alternation bound). *Let P be a point set with distinct y-coordinates, and let $\mathcal{T}$ be a binary search tree over the values $P.x$. We define $\mathsf{Alt}_{\mathcal{T}}(P)$ using the recursive structure of $\mathcal{T}$. If $\mathcal{T}$ is a single node, let $\mathsf{Alt}_{\mathcal{T}}(P) := 0$. Otherwise, let $\mathcal{T}_{\mathtt{L}}$ and $\mathcal{T}_{\mathtt{R}}$ be the left and right subtrees at the root. Partition P into two sets $P_{\mathtt{L}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{L}}\}$ and $P_{\mathtt{R}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{R}}\}$ and consider the quantity $\mathrm{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y)$, which describes how much $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ are interleaved with time (we call each switch between $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ a "preferred child alternation"). Then*

$$\mathsf{Alt}_{\mathcal{T}}(P) := \mathrm{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}}(P_{\mathtt{L}}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}}(P_{\mathtt{R}}). \tag{4}$$

The alternation bound is then defined as the maximum over all trees:

$$\mathsf{Alt}(P) := \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(P).$$

In addition, for an access sequence X, let $\mathsf{Alt}_{\mathcal{T}}(X) := \mathsf{Alt}_{\mathcal{T}}(G_X)$ and $\mathsf{Alt}(X) := \mathsf{Alt}(G_X)$.

For the reason of space, we describe Wilber's Funnel bounds in the full version.

▶ **Definition 14** (amortized bounds). *For any sequence $X \in S^m$, define amortized versions of the optimal cost and the Wilber bounds:*

$$\overline{\mathsf{OPT}}(X) := \frac{\mathsf{OPT}(X)}{m}, \quad \overline{\mathsf{Alt}}(X) := \frac{\mathsf{Alt}(X)}{m}, \quad \overline{\mathsf{Funnel}}(X) := \frac{\mathsf{Funnel}(X)}{m}.$$

▶ **Definition 15** (composed sequence, see [9]). *Let $S_1, \dots, S_l$ be disjoint sets of keys with increasing values (i.e. $\forall x \in S_j, x' \in S_{j+1}$, we have $x < x'$). For each $j \in [l]$, let $X^{(j)} \in S_j^{m_j}$ be an access sequence with keys in S_j, and let $\widetilde{X}$ be a sequence with keys in $[l]$ such that every $j \in [l]$ appears exactly m_j times (its total length is $m := m_1 + \cdots + m_l$). Then we define the composed sequence*

$$X = \widetilde{X}(X^{(1)}, \dots, X^{(l)}) \in (S_1 \cup \cdots \cup S_l)^m$$

as the sequence that interleaves $X^{(1)}, \dots, X^{(l)}$ according to the order given by $\widetilde{X}$: that is, $X_t = X_{\sigma(t)}^{(\widetilde{X}_t)}$ where $\sigma(t) := \#\left\{t' \leq t \,\middle|\, \widetilde{X}_{t'} = \widetilde{X}_t\right\}$.

[6] These definitions may differ by a constant factor or an additive $\pm O(m)$ from the definitions the reader has seen before. We will ignore such differences, because the cost of a BST also varies by $\pm O(m)$ depending on the definition, and the interesting regime is when $\mathsf{OPT}(X) = \omega(m)$.

Note that [6] defines the *decomposition* operation, which is the inverse operation of the composition. We will use Definition 15 throughout this paper.

▶ **Definition 16** (j_x). *In the context of Definition 15, for any key $x \in S_1 \cup \cdots \cup S_l$, let j_x be the unique index such that $x \in S_{j_x}$.*

3 Effect of Composition on Wilber's bounds

We prove Theorem 1 in this section. Namely, we show that Wilber's bounds act nicely under composition, allowing us to boost the separation between them in Section 4. We divide the proofs into the following two theorems.

▶ **Theorem 17** (subadditivity of Alt). *Let $X := \widetilde{X}(X^{(1)}, \ldots, X^{(l)})$ be a composed sequence with $|X^{(1)}| = \cdots = |X^{(l)}|$.*[7] *Then*

$$\mathsf{Alt}(X) \leq \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).$$

▶ **Theorem 18** (superadditivity of Funnel). *Let $X := \widetilde{X}(X^{(1)}, \ldots, X^{(l)})$ be a composed sequence. Then*

$$\mathsf{Funnel}(X) \geq \mathsf{Funnel}\left(\widetilde{X}\right) + \sum_j \mathsf{Funnel}\left(X^{(j)}\right) - O(m).$$

For the reason of space, we postpone the proof of Theorem 18 to the full version.

3.1 Subadditivity of the alternation bound

We prove Theorem 17 in this section.

Proof plan

We will show that for any binary search tree $\mathcal{T}$ over $S_1 \cup \cdots \cup S_l$,

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).$$

We will do this by

- decomposing $\mathcal{T}$ into the corresponding binary trees $\widetilde{\mathcal{T}}$ over $[l]$ and $\mathcal{T}_j$ over S_j for all j;
- classifying preferred child alternations in $\mathcal{T}$ into 4 types, which correspond to either
  - alternations in $\widetilde{\mathcal{T}}$,
  - alternations in $\mathcal{T}_j$ for some j,
  - or to some other events that happen at most $O(m)$ times in aggregate.

That is, we will show that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m).$$

[7] We make this assumption so that the proof is simpler.

3.1.1 Decomposing the tree

For a tree $\mathcal{T}$, we write $x \prec_{\mathcal{T}} b$ if x is a descendent of b in $\mathcal{T}$, and we write $S \prec_{\mathcal{T}} b$ if $x \prec_{\mathcal{T}} b$ for all $x \in S$.

▶ **Definition 19.** *Let $\mathcal{T}_j$ be the unique binary search tree over S_j such that if $b, x \in S_j$ and $x \prec_{\mathcal{T}} b$ then $x \prec_{\mathcal{T}_j} b$.*

$\mathcal{T}_j$ is constructed by running the following recursive algorithm, which builds a tree $\mathcal{T}_{\text{out}}$:

- Start at the root of $\mathcal{T}$, and let x be the current node.
- If $x \in S_j$, then
  - make x the root of $\mathcal{T}_{\text{out}}$;
  - form x's left subtree in $\mathcal{T}_{\text{out}}$ by recursing on x's left subtree in $\mathcal{T}$;
  - form x's right subtree in $\mathcal{T}_{\text{out}}$ by recursing on x's right subtree in $\mathcal{T}$.
- If $x \notin S_j$ then since S_j is contiguous, at most one of x's left and right subtrees in $\mathcal{T}$ can contain elements from S_j.
  - If there is one such subtree, form $\mathcal{T}_{\text{out}}$ by recursing on it.
  - Otherwise let $\mathcal{T}_{\text{out}}$ be the empty tree.

This algorithm clearly has the desired properties:

- Clearly, by construction, $\mathcal{T}_j$ is a binary search tree and its set of keys is S_j.
- If $b, x \in S_j$ and $x \prec_{\mathcal{T}} b$, then $x \prec_{\mathcal{T}_j} b$, because the only way to get to x is to first pass through b, add it as a root of the current subtree, then recurse on b's subtree that contains x, which eventually adds x to $\mathcal{T}_j$ as a descendent of b.
- $\mathcal{T}_j$ is unique since the only nontrivial choice the algorithm makes is to add x as a root, but this is necessary since it must be an ancestor of all of the keys that later get added to this part of $\mathcal{T}_j$.

▶ **Definition 20.** *Let $\widetilde{\mathcal{T}}$ be the unique binary search tree over $[l]$ such that if $x \prec_{\mathcal{T}} b$ and $S_{j_b}, S_{j_x} \prec_{\mathcal{T}} b$ then $j_x \prec_{\widetilde{\mathcal{T}}} j_b$.*

$\widetilde{\mathcal{T}}$ is constructed by running the following recursive algorithm, which builds a tree $\mathcal{T}_{\text{out}}$:

- Start at the root of $\mathcal{T}$, and let x be the current node.
- If j_x hasn't already been seen earlier in the algorithm (which happens iff x is the lowest common ancestor of all of S_{j_x} in $\mathcal{T}$), then
  - make j_x the root of $\mathcal{T}_{\text{out}}$;
  - form j_x's left subtree in $\mathcal{T}_{\text{out}}$ by recursing on x's left subtree in $\mathcal{T}$;
  - form j_x's right subtree in $\mathcal{T}_{\text{out}}$ by recursing on x's right subtree in $\mathcal{T}$.
- If j_x has already been seen earlier in the algorithm, then some ancestor of x was also in S_{j_x}, and S_{j_x} is contiguous, so S_{j_x} must contain either x's entire left subtree, or x's entire right subtree. That means that at most one of x's subtrees can contain elements *not* in S_{j_x}.
  - If there is one such subtree, form $\mathcal{T}_{\text{out}}$ by recursing on it.
  - Otherwise, let $\mathcal{T}_{\text{out}}$ be the empty tree.

This algorithm clearly has the desired properties:

- Clearly, by construction, $\widetilde{\mathcal{T}}$ is a binary search tree and its set of keys is $[l]$.
- If $x \prec_{\mathcal{T}} b$ and $S_{j_b}, S_{j_x} \prec_{\mathcal{T}} b$ then:
  - We can assume $j_x \neq j_b$ and thus $x \neq b$, otherwise the claim is trivially true.
  - Since $S_{j_b} \prec_{\mathcal{T}} b$, b must be the lowest common ancestor of all of S_{j_b} in $\mathcal{T}$, so j_b gets added to $\widetilde{\mathcal{T}}$ when the algorithm is looking at node b.

  - Also, since $S_{j_x} \prec_{\mathcal{T}} b$ and $j_x \neq j_b$, that means that all the elements from S_{j_x} are descendents of b, so j_x will be added to $\widetilde{\mathcal{T}}$ in one of the recursive branches launched when looking at b.
  - Therefore j_x will be a descendent of j_b in $\widetilde{\mathcal{T}}$.
- $\widetilde{\mathcal{T}}$ is unique since the only nontrivial choice the algorithm makes is to add j_x as root when it first sees an element from S_{j_x}, but this is necessary since for any j which will eventually be added to this part of the tree, S_j must have been completely contained in x's subtree, and therefore j_x must be an ancestor of j.

3.1.2 Stating the classification

Consider some left-to-right[8] preferred child alternation that X produces in $\mathcal{T}$. That is, take some value $b \in S_1 \cup \cdots \cup S_l$ and some times $t_x < t_y \in [m]$ such that

- $x := X_{t_x}$ is in the left subtree of b in $\mathcal{T}$,
- $y := X_{t_y}$ is in the right subtree of b in $\mathcal{T}$,
- and none of the accesses $X_{t_x+1}, \ldots, X_{t_y-1}$ made in the interim were to values that are strict descendents of b.

Let a be the lowest ancestor of b such that $a < b$ and c be the lowest ancestor of b such that $b < c$.[9] This means that the left and right subtrees of b correspond to the keys in intervals (a, b) and (b, c). We have $x, y \prec_{\mathcal{T}} b$, $x \in (a, b)$, and $y \in (b, c)$.

▷ Claim 21. One of the following must hold (from most "local" to most "global"):

1. All of b, x, y are in the same range S_{j_b}, so x and y are in the left and right subtrees of b in $\mathcal{T}_{j_b}$, and this corresponds to an alternation in $\mathcal{T}_{j_b}$.
2. b is either the highest ancestor of x such that $b \in S_{j_x}$ and $x < b$, or the highest ancestor of y such that $b \in S_{j_y}$ and $b < y$.
3. $S_{j_b} \prec_{\mathcal{T}} b$, and either j_x is the closest (in key value) ancestor of j_b in $\widetilde{\mathcal{T}}$ such that $j_x < j_b$, or j_y is the closest (in key value) ancestor of j_b in $\widetilde{\mathcal{T}}$ such that $j_b < j_y$.
4. All of b, x, y are in different ranges (i.e. $j_x < j_b < j_y$), j_x is in j_b's left subtree in $\widetilde{\mathcal{T}}$, and j_y is in j_b's right subtree in $\widetilde{\mathcal{T}}$, so this corresponds to an alternation in $\widetilde{\mathcal{T}}$.

3.1.3 Proving the classification

Proof of Claim 21. Let us prove that every alternation is of one of these four types.

- First, suppose that $j_x = j_b = j_y$. Then by construction of $\mathcal{T}_{j_b}$, x and y are still descendents of b in $\mathcal{T}_{j_b}$, and since $\mathcal{T}_{j_b}$ is a binary search tree, x must be in b's left subtree and y must be in b's right subtree. So this is type 1.
- Now suppose that exactly one of $j_x = j_b$ and $j_b = j_y$ holds. By symmetry, suppose that it is the former, and thus $j_x = j_b < j_y$. Then we trivially have $b \in S_{j_x}$. And on the other hand, consider any ancestor b' of x which is higher than b and satisfies $x < b'$. Then b' would have to satisfy $y < b'$ as well, and in particular $j_y \leq j_{b'}$, so it could not lie in S_{j_x}. Therefore b is the highest ancestor of x which lies in S_{j_x} and satisfies $x < b$. So this is type 2.

[8] The case where the alternation occurs from right to left is analogous.

[9] If either a or c doesn't exist, let $a := -\infty$ or $c := +\infty$ by convention.

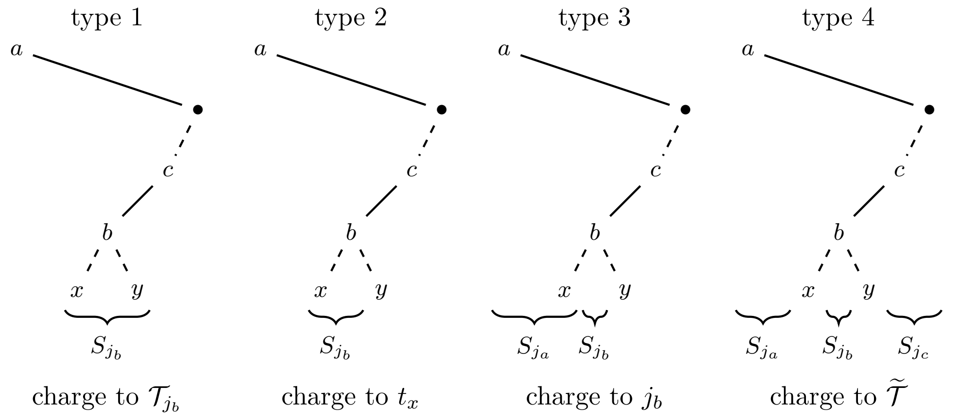


Figure 4 A preferred child alternation in $\mathcal{T}$: b's preferred child changes from the left side (due to an access to x) to the right side (due to an access to y). There are four qualitatively different ways in which the alternation can happen depending on which ranges $S_1, \ldots, S_l$ the keys a, b, c, x, y belong.

- From now on we can assume that $j_x < j_b < j_y$, which means in particular that $j_a < j_b < j_c$, that S_{j_b} is contained entirely in b's subtree in $\mathcal{T}$, and therefore b is the highest member of S_{j_b} in $\mathcal{T}$. Now, the lowest common ancestor of S_{j_a} (resp. S_{j_c}) must be an ancestor of a (resp. c) and therefore an ancestor of b, so by the properties of $\widetilde{\mathcal{T}}$, j_a (resp. j_c) is an ancestor of j_b in $\widetilde{\mathcal{T}}$. Furthermore, any ancestor of j_b in $\widetilde{\mathcal{T}}$ must be of the form j_z for some ancestor z of b in $\mathcal{T}$, so since a (resp. c) is the closest (in key value) ancestors of b on its left (resp. right) side in $\mathcal{T}$, j_a (resp. j_c) must be the closest ancestor of j_b on its left (resp. right) side in $\widetilde{\mathcal{T}}$.
  - Suppose that at least one of $j_a = j_x$ or $j_y = j_c$ holds. By symmetry suppose that it is the former. Then just by virtue of the fact that $j_x = j_a$, j_x is the closest ancestor of j_b on its left side in $\widetilde{\mathcal{T}}$. So this is type 3.
  - Otherwise, we have $j_a < j_x < j_b < j_y < j_c$. This implies that S_{j_x} lies entirely within b's left subtree, and S_{j_y} lies entirely within b's right subtree, thus j_x and j_y are descendents of j_b in $\widetilde{\mathcal{T}}$. So this is type 4. ◀

3.1.4 Using the classification to prove Theorem 17

Proof of Theorem 17. Let $\mathcal{T}$ be any binary search tree over $S_1 \cup \cdots \cup S_l$, and let $\mathcal{T}_j$ and $\widetilde{\mathcal{T}}$ be the corresponding trees defined in Definition 19 and Definition 20. We will show that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m).$$

Let us use Claim 21: we charge type 1 alternations to $\mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right)$, type 4 alternations to $\mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right)$, and we show below that there are only $O(m)$ alternations of types 2 and 3.

For type 2, this is because we can charge it uniquely to the access made to x or y (formally, we charge it to t_x or t_y).

- Let us take the first subcase: b is the highest ancestor of x such that $b \in S_{j_x}$ and $x < b$. x can only have one highest ancestor with a given property, so it has only one highest "ancestor b such that $b \in S_{j_x}$ and $x < b$". So this can apply to at most one of the alternations that occurred when accessing x, and thus we can charge it to t_x.

- Let us take the second subcase: b is the highest ancestor of y such that $b \in S_{j_y}$ and $b < y$. Again, y can only have one highest "ancestor b such that $b \in S_{j_y}$ and $b < y$". The access to x is the first time that the preferred child switches back from b's left child to b's right child after accessing y. So this event is unique from the perspective of this particular access to y, and thus we can charge it to t_y.

Finally, we will bound the total number of occurrences of type 3 alternations by charging them to j_b, not uniquely but in a $\frac{l}{m}$-to-1 manner. Let us take the case where j_x is the closest ancestor of j_b in $\widetilde{\mathcal{T}}$ such that $j_x < j_b$ (the other case is analogous). Clearly, j_b determines j_x uniquely. And since b's subtree contains S_{j_b} in its entirety, j_b determines b uniquely too. So once you know j_b, the only uncertainty left about this alternation is *which* access within $X^{(j_x)}$ caused it. So the total number of alternations of this type is bounded by

$$\underbrace{l}_{\text{which } j_b?} \qquad \underbrace{\max_{j_x}\left|X^{(j_x)}\right|}_{\text{which access within } X^{(j_x)}?} \qquad = l\frac{m}{l} = m,$$

where the first equality uses our assumption that $|X^{(1)}| = \cdots = |X^{(l)}|$.

Overall, we have shown that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \underbrace{\mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right)}_{\text{type 4}} + \sum_j \underbrace{\mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right)}_{\text{type 1}} + \underbrace{O(m)}_{\text{type 2 (charge to } t_x \text{ or } t_y)} + \underbrace{O(m)}_{\text{type 3 (charge to } j_b)},$$

so we can now take the maximum over $\mathcal{T}$ to conclude

$$\begin{aligned}\mathsf{Alt}(X) &:= \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(X)\\ &\leq \max_{\mathcal{T}}\left(\mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m)\right)\\ &\leq \max_{\widetilde{\mathcal{T}}} \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \max_{\mathcal{T}_j} \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m)\\ &= \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).\end{aligned}$$

◀

4 Boosting the separation between Wilber's bounds

We prove Theorem 2 in this section. We now use the composition properties of Alt and Funnel we proved in Section 3 to show that Tango tree makes an optimal trade-off between fixed costs and variable costs that depend on the alternation bound.

4.1 What boosting can we get?

Lecomte and Weinstein [27] show an $\Omega(\log \log n)$ separation between Alt and Funnel.

▶ **Theorem 22** (Theorem 2 in [27]). *For any n of the form 2^{2^k}, there is a sequence $Y_n \in [n]^m$ where $m \leq \text{poly}(n)$, each element appears $O(m/n)$ times, and*

$$\begin{aligned}\overline{\mathsf{Alt}}(Y_n) &\leq O(1)\\ \overline{\mathsf{Funnel}}(Y_n) &\geq \Omega(\log\log n).\end{aligned}$$

We can use the tight composition results from Section 3 to show the following boosted separation. We emphasize that the approximate subadditivity of the Alternation bound [6] is insufficient to boost the separation.

▶ **Theorem 23** (Hardness Amplification, Restatement of Theorem 2)**.** *There is a constant $K > 0$ such that for any n of the form 2^{2^k} and any power-of-two $R \le \frac{\log n}{K}$, there is a sequence $Y_n^{\circ R} \in [n]^{m'}$ with $m' \le \operatorname{poly}(n)$ such that*

$$\begin{aligned}\overline{\mathsf{Alt}}\left(Y_n^{\circ R}\right) &\le O(R)\\ \overline{\mathsf{Funnel}}\left(Y_n^{\circ R}\right) &\ge \Omega\left(R\log\left(\frac{\log n}{R}\right)\right).\end{aligned}$$

Proof. Let Y_n be the sequence stated in Theorem 22. First, pad Y_n so that each key appears *exactly* m/n times, by adding each key one by one the appropriate number of times, in ascending order. It is easy to see that this maintains the bounds

$$\begin{aligned}\overline{\mathsf{Alt}}(Y_n) &\le O(1)\\ \overline{\mathsf{Funnel}}(Y_n) &\ge \Omega(\log\log n).\end{aligned}$$

Now, let $C_O, C_\Omega > 0$ be constants such that

$$\begin{aligned}\overline{\mathsf{Alt}}(Y_n) &\le C_O\\ \overline{\mathsf{Funnel}}(Y_n) &\ge C_\Omega \log\log n\end{aligned}$$

(we will allow ourselves to make C_O even larger later on).

Let $Y_n^{\circ 1} := Y_n$, and for all power-of-two $R \ge 1$, let

$$Y_n^{\circ 2R} := \left(Y_{\sqrt{n}}^{\circ R}\right)^{\otimes\sqrt{n}}\left(Y_{\sqrt{n}}^{\circ R},\dots,Y_{\sqrt{n}}^{\circ R}\right),$$

where "$X^{\otimes\sqrt{n}}$" means "X repeated $\sqrt{n}$ times", and with an abuse of notation we assume that the $\sqrt{n}$ sequences $Y_{\sqrt{n}}^{\circ R},\dots,Y_{\sqrt{n}}^{\circ R}$ that are being composed each contains a distinct range of keys. We can check that

$$\left|Y_n^{\circ R}\right| = n^{1-1/R}|Y_{n^{1/R}}| \le n^{1-1/R}\operatorname{poly}\left(n^{1/R}\right) \le \operatorname{poly}(n)$$

as desired. We will show by induction that

$$\begin{aligned}\overline{\mathsf{Alt}}\left(Y_n^{\circ R}\right) &\le C_O(2R-1)\\ \overline{\mathsf{Funnel}}\left(Y_n^{\circ R}\right) &\ge C_\Omega\frac{R+1}{2}\log\left(\frac{\log n}{R}\right).\end{aligned}$$

Base case: $R = 1$. We verify that indeed

$$\begin{aligned}\overline{\mathsf{Alt}}(Y_n) &\le C_O\\ &= C_O(2\cdot 1-1)\\ &= C_O(2R-1)\end{aligned}$$

and

$$\begin{aligned}\overline{\mathsf{Funnel}}(Y_n) &\ge C_\Omega\log\log n\\ &= C_\Omega\frac{1+1}{2}\log\left(\frac{\log n}{1}\right)\\ &= C_\Omega\frac{R+1}{2}\log\left(\frac{\log n}{R}\right).\end{aligned}$$

Inductive case: $R \to 2R$. Suppose this is true for some $R \geq 1$, for all n. Then for Alt, by Theorem 17 we have

$$\begin{aligned}\overline{\mathsf{Alt}}\left(Y_n^{\circ 2R}\right) &\leq \overline{\mathsf{Alt}}\left(Y_{\sqrt{n}}^{\circ R}\right) + \frac{\sqrt{n} \cdot \overline{\mathsf{Alt}}\left(Y_{\sqrt{n}}^{\circ R}\right)}{\sqrt{n}} + O(1) \\ &\leq 2C_O(2R-1) + O(1) \\ &= C_O(4R-1) - C_O + O(1) \\ &\leq C_O(4R-1),\end{aligned}$$

where the last step holds as long as C_O is large enough.

For Funnel, by Theorem 18 we have

$$\begin{aligned}\overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right) &\geq \overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right) + \frac{\sqrt{n} \cdot \overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right)}{\sqrt{n}} - O(1) \\ &\geq C_\Omega(R+1)\log\left(\frac{\log\sqrt{n}}{R}\right) - O(1) \\ &= C_\Omega\frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right) + \frac{C_\Omega}{2}\log\left(\frac{\log n}{2R}\right) - O(1) \\ &\geq C_\Omega\frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right) + \frac{C_\Omega}{2}\log\left(\frac{K}{2}\right) - O(1) \\ &\geq C_\Omega\frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right),\end{aligned}$$

where the penultimate step holds because $R \leq \frac{\log n}{K}$, and the last step holds as long as K is large enough. ◀

5 Optimality of Tango Trees

We are ready to prove Theorem 3.

▶ **Theorem 24** (Restatement of Theorem 3). *Let $\alpha, \beta : \mathbb{N} \to \mathbb{R}_{\geq 1}$ be any functions and let A be some BST algorithm. Denote the amortized cost of A on an access sequence X as $\overline{A}(X)$. Suppose that for all access sequences X over n keys, we have*

$$\overline{A}(X) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n).$$

Then $\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$ for infinitely many values of n.

Proof. In fact we will show that the result holds under the weaker assumption that the theorem holds for the *optimal* amortized cost:

$$\overline{\mathsf{OPT}}(X) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n).$$

The above inequality must in particular hold for the access sequence $Y_n^{\circ R}$ from Theorem 2, where we let R be the largest power of two such that $R \leq \beta(n)$. This gives us

$$\begin{cases}\overline{\mathsf{OPT}}\left(Y_n^{\circ R}\right) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n) \leq O(R(\alpha(n)+1)) \\ \overline{\mathsf{OPT}}\left(Y_n^{\circ R}\right) \geq \Omega\left(\overline{\mathsf{Funnel}}\left(Y_n^{\circ R}\right)\right) \geq \Omega\left(R\log\left(\frac{\log n}{R}\right)\right).\end{cases}$$

Combining these inequalities, we obtain

$$R(\alpha(n)+1) \geq \Omega\left(R\log\left(\frac{\log n}{R}\right)\right) \implies \alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{R}\right)\right) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$$

(where the implication uses the fact that $\alpha(n) \geq 1$). ◀

References

1 Brian Allen and Ian Munro. Self-organizing binary search trees. *J. ACM*, 25(4):526–535, October 1978. doi:10.1145/322092.322094.

2 Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. Direct sums in randomized communication complexity. *Electron. Colloquium Comput. Complex.*, TR09-044, 2009. URL: https://eccc.weizmann.ac.il/report/2009/044, arXiv:TR09-044.

3 Benjamin Aram Berendsohn, László Kozma, and Michal Opler. Optimization with pattern-avoiding input. *CoRR*, abs/2310.04236, 2023. doi:10.48550/arXiv.2310.04236.

4 Guy E. Blelloch and Magdalen Dobson. The geometry of tree-based sorting. In *ICALP*, volume 261 of *LIPIcs*, pages 26:1–26:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.26.

5 Prosenjit Bose, Karim Douïeb, Vida Dujmovic, and Rolf Fagerberg. An $O(\log \log n)$-competitive binary search tree with optimal worst-case access times. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 38–49, 2010. doi:10.1007/978-3-642-13731-0_5.

6 Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the strong wilber 1 bound for binary search trees. In *APPROX-RANDOM*, volume 176 of *LIPIcs*, pages 33:1–33:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.33.

7 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *WADS*, volume 9214 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2015. doi:10.1007/978-3-319-21840-3_13.

8 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 410–423. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.32.

9 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv preprint*, 2016. arXiv:1603.04892.

10 Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy bsts via matrix decomposition. In *SODA*, pages 509–534. SIAM, 2023. doi:10.1137/1.9781611977554.CH22.

11 Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Akash Pareek, and Sorrachai Yingchareonthawornchai. The group access bounds for binary search trees. *CoRR*, abs/2312.15426, 2023. doi:10.48550/arXiv.2312.15426.

12 Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *ESA*, volume 173 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.28.

13 Parinya Chalermsook, Seth Pettie, and Sorrachai Yingchareonthawornchai. Sorting pattern-avoiding permutations via 0-1 matrices forbidding product patterns. In *SODA*, pages 133–149. SIAM, 2024. doi:10.1137/1.9781611977912.7.

14 Richard Cole. On the dynamic finger conjecture for splay trees. part II: the proof. *SIAM J. Comput.*, 30(1):44–85, 2000. doi:10.1137/S009753979732699X.

15 Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825, doi:10.1137/1.9781611973068.55.

16 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.

17 Jonathan Derryberry, Daniel Dominic Sleator, and Chengwen Chris Wang. A lower bound framework for binary search trees with rotations. *Technical report*, 2005.

18 Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theor. Comput. Sci.*, 314(3):459–466, 2004. doi:10.1016/J.TCS.2004.01.019.

19 Kyle Fox. Upper bounds for maximally greedy binary search trees. In *WADS*, volume 6844 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 2011. doi:10.1007/978-3-642-22300-6_35.

20 Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's xor-lemma. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 273–301. Springer, 2011. doi:10.1007/978-3-642-22670-0_23.

21 Navin Goyal and Manoj Gupta. Better analysis of greedy binary search tree on decomposable sequences. *Theor. Comput. Sci.*, 776:19–42, 2019. doi:10.1016/J.TCS.2018.12.021.

22 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 236–250, 2013. doi:10.1007/978-3-642-40273-9_16.

23 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Comput. Complex.*, 5(3/4):191–204, 1995. doi:10.1007/BF01206317.

24 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 197–208. ACM, 2021. doi:10.1145/3406325.3450999.

25 László Kozma. Binary search trees, rectangles and patterns, 2016.

26 László Kozma and Thatchaphol Saranurak. Smooth heaps and a dual view of self-adjusting data structures. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 801–814, 2018. doi:10.1145/3188745.3188864.

27 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. In *ESA*, volume 173 of *LIPIcs*, pages 68:1–68:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.68.

28 Troy Lee, Adi Shraibman, and Robert Spalek. A direct product theorem for discrepancy. In *CCC*, pages 71–80. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.25.

29 Caleb C. Levy and Robert E. Tarjan. A new path from splay to dynamic optimality. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1311–1330. SIAM, 2019. doi:10.1137/1.9781611975482.80.

30 Joan Marie Lucas. *Canonical forms for competitive binary search tree algorithms*. Rutgers University, Department of Computer Science, Laboratory for Computer Science Research, 1988.

31 J. Ian Munro. On the competitiveness of linear search. In Mike Paterson, editor, *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer, 2000. doi:10.1007/3-540-45253-2_31.

32 Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1115–1124, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347204.

33 Mihai Puatracscu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM J. Comput.*, 39(2):730–741, 2009. doi:10.1137/070684859.

34 Ran Raz. A counterexample to strong parallel repetition. *SIAM J. Comput.*, 40(3):771–777, 2011. doi:10.1137/090747270.

35 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Comb.*, 19(3):403–435, 1999. doi:10.1007/S004930050062.

36 Yaniv Sadeh and Haim Kaplan. Dynamic binary search trees: Improved lower bounds for the greedy-future algorithm. In *STACS*, volume 254 of *LIPIcs*, pages 53:1–53:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.53.

37 Ronen Shaltiel. Towards proving strong direct product theorems. In *Proceedings of the 16th Annual Conference on Computational Complexity*, CCC '01, page 107, USA, 2001. IEEE Computer Society.

38 Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM J. Comput.*, 41(5):1122–1165, 2012. doi:10.1137/110842661.

39 Yaroslav Shitov. A counterexample to comon's conjecture. *SIAM J. Appl. Algebra Geom.*, 2(3):428–443, 2018. doi:10.1137/17M1131970.

40 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985. doi:10.1145/3828.3835.

41 Rajamani Sundar. On the deque conjecture for the splay algorithm. *Comb.*, 12(1):95–124, 1992. doi:10.1007/BF01191208.

42 Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109600.

43 R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, February 1989. doi:10.1137/0218004.

Reconfiguration of Labeled Matchings in Triangular Grid Graphs

Naonori Kakimura
Department of Mathematics, Keio University, Yokohama, Japan

Yuta Mishima
The Japan Research Institute, Limited, Tokyo, Japan

Abstract

This paper introduces a new reconfiguration problem of matchings in a triangular grid graph. In this problem, we are given a nearly perfect matching in which each matching edge is labeled, and aim to transform it to a target matching by sliding edges one by one. This problem is motivated to investigate the solvability of a sliding-block puzzle called "Gourds" on a hexagonal grid board, introduced by Hamersma et al. [ISAAC 2020]. The main contribution of this paper is to prove that, if a triangular grid graph is factor-critical and has a vertex of degree 6, then any two matchings can be reconfigured to each other. Moreover, for a triangular grid graph (which may not have a degree-6 vertex), we present another sufficient condition using the local connectivity. Both of our results provide broad sufficient conditions for the solvability of the Gourds puzzle on a hexagonal grid board with holes, where Hamersma et al. left it as an open question.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases combinatorial reconfiguration, matching, factor-critical graphs, sliding-block puzzles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.43

Related Version *arXiv Version*: `https://arxiv.org/abs/2409.11723` [19]

Funding *Naonori Kakimura*: Supported by JSPS KAKENHI Grant Numbers JP22H05001, JP20H05795, and 23K21646, Japan and JST ERATO Grant Number JPMJER2301, Japan.

Acknowledgements The authors are grateful to Yushi Uno for valuable discussions. We are also grateful for the anonymous referees of ISAAC 2024 for their helpful comments.

1 Introduction

Combinatorial reconfiguration is a fundamental research subject that studies the solution space of combinatorial problems. A typical example is solving sliding-block puzzles such as the 15-puzzle. The 15-puzzle can be viewed as the transformation between the arrangement of puzzle pieces, and the goal is to transform an initial arrangement of pieces to a given target arrangement. Combinatorial reconfiguration has applications in a variety of fields such as mathematical puzzles, operations research, and computational geometry. See the surveys by Nishimura [23] and van den Heuvel [28].

Hamersma et al. [13] introduced a new sliding-block puzzle on a hexagonal grid, which they call *Gourds*. The name "gourd" refers to the shape of the puzzle pieces, which are essentially 1×2 pieces on a board. Like in the 15-puzzle, only one grid cell is empty. The goal is to obtain a target configuration of pieces on the board by moving pieces one-by-one, similar to the 15-puzzle. Here we allow a piece to make three different kinds of moves: slide, turn, and pivot (see [13] for the details). They characterized hexagonal grid boards without holes such that the Gourds puzzle[1] is always solvable, and left it as a main open question to characterize boards *with holes*.

[1] In this paper, the Gourds puzzle refers to the numbered type in [13] where each piece has numbers.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 43; pp. 43:1–43:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Motivated by the study of the Gourds puzzle, we introduce a reconfiguration problem of matchings in a triangular grid graph. In the problem, we are given a matching that exposes only one vertex, which is called a *nearly perfect matching.* Each matching edge, which corresponds to a puzzle piece, is labeled. We are allowed to slide a matching edge toward the exposed vertex. The goal is to move matching edges one-by-one to obtain a target labeled matching. It should be emphasized that each edge in the given matching has to be moved to the edge with the same label in the target matching. See Section 2 for the formal definition. We remark that our problem can be defined on a general graph, which may be of independent interest. The problem setting is different from the matching reconfiguration problems studied in the literature. See Section 1.1.

In this paper, we investigate the reconfigurability of the above reconfiguration problem of labeled matchings on a triangular grid graph. In particular, we aim to characterize a triangular grid graph such that any two labeled matchings can be reconfigured to each other. We call such a graph *reconfigurable.*

As mentioned in Section 3, it is not difficult to observe that, if a graph is reconfigurable, then it is 2-connected and factor-critical. A graph is *factor-critical* if it has a nearly perfect matching that exposes any vertex. These two conditions, however, are not sufficient, as there exists a 2-connected factor-critical graph that is not reconfigurable.

The main contribution of this paper is to prove that, if a 2-connected factor-critical triangular grid graph has at least one vertex of degree 6, then it is reconfigurable. Our results can be adapted to the Gourds puzzle by taking the dual of a triangular grid graph, which implies that the Gourds puzzle can always be solved when at least one hexagonal cell on the board does not touch the holes or the outer face.

The key idea to prove the main result is to exploit the ear decomposition in matching theory. A factor-critical graph is known to have a constructive characterization using ear decomposition with odd paths and cycles. Using the ear structure, we show that, if an ear decomposition starts from a reconfigurable subgraph, then we can recursively find reconfiguration steps between any two labeled matchings. However, every ear decomposition does not necessarily satisfy the above assumption. We then investigate the matching structure of a triangular grid graph to identify simple reconfigurable subgraphs such that every triangular grid graph with a vertex of degree 6 admits an ear decomposition starting from one of them.

In addition, for a triangular grid graph (which may not have a vertex of degree 6), we present another sufficient condition for the reconfigurability using the local connectivity. A graph is said to be *locally-connected* if the neighbor vertices of any vertex induce a connected graph. We prove that, if a triangular grid graph is locally-connected, but not isomorphic to the Star of David graph (Figure 1), then it is reconfigurable. Moreover, we show that, for a graph with $2n + 1$ vertices, we can find in polynomial time reconfiguration steps with length $O(n^3)$.

The characterization for the Gourds puzzle by Hamersma et al. [13] implies that a triangular grid graph is reconfigurable if it is 2-connected, but not isomorphic to the Star of David graph, and has no *holes*, where a hole is a face with boundary length at least 6. Our conditions, which allow to have a hole, are much broader than theirs, as the local connectivity and the 2-connectivity are equivalent for a graph with no holes.

Due to the space limitation, we omit the proofs of statements with the symbol $\star$ marks, which may be found in the full version of this paper [19].

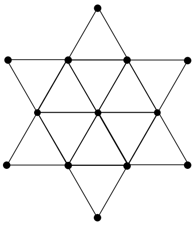


Figure 1 The Star of David graph.

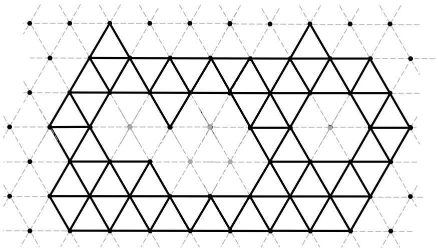


Figure 2 A triangular grid graph with 2 holes.

1.1 Related Work

A factor-critical graph plays an important role in matching theory. It is known that any non-bipartite graph can be decomposed in terms of maximum matching, called the *Gallai-Edmonds* decomposition. It essentially decomposes a given graph into factor-critical graphs, graphs with perfect matchings, and bipartite graphs. Also, factor-critical graphs are used to describe facets of the matching polytope of a given graph. See [21, 26] for the details.

Sliding-block puzzles have been investigated in both recreational mathematics and algorithms research. The 15-puzzle was introduced as a prize problem by Sam Loyd in 1878 [27]. In the 15-puzzle, it is characterized by odd/even permutations whether any configuration can be realized or not [18]. However, it is NP-complete for $n \times n$ boards to compute the smallest number of steps to reach a given configuration [10, 24]. There are many variants of the 15-puzzle such as Rush Hour [6, 11] and rolling-block puzzles [7]. Many puzzles have been shown NP-hard or PSPACE-hard (see e.g., [15]).

In the literature of combinatorial reconfiguration, the reconfiguration of matchings has been studied extensively. Ito et al. [16] initiated to study a reconfiguration problem of matchings. The aim is to decide whether a given matching can be transformed to a target matching by adding/removing a matching edge in each step. They showed that the problem can be solved in polynomial time with the aid of the Gallai-Edmonds decomposition. On reconfigurating perfect matchings, Ito et al. [17] studied the shortest transformation of perfect matchings by taking the symmetric difference along an alternating cycle, motivated by the study of a diameter of the matching polytope (see also [8, 25]). Bonamy et al. [3] restricts the length of alternating cycles used in the transformations to be 4. We remark that all the above mentioned problems aim to transform an initial (perfect) matching to a target one in which their matching edges are *not* labeled.

By taking the line graph of a given graph, reconfiguration problems of matchings can be viewed as reconfiguration problems of independents sets. Our problem setting is related to its variant known as the token sliding problem. The token sliding problem is PSPACE-complete, even on restricted graph classes such as planar graphs [14]. On the other hand, the problem can be solved in linear time on trees [9], and it is fixed-parameter tractable on bounded degree graphs [2]. See also [5] for the survey on the independent set and dominating set reconfiguration problems. Another related problem is the token swapping problem. In the problem, we are given tokens on each vertex of a graph, and we want to move every token to its target position by swapping two adjacent tokens. See e.g., [1, 4, 20, 22] and references therein.

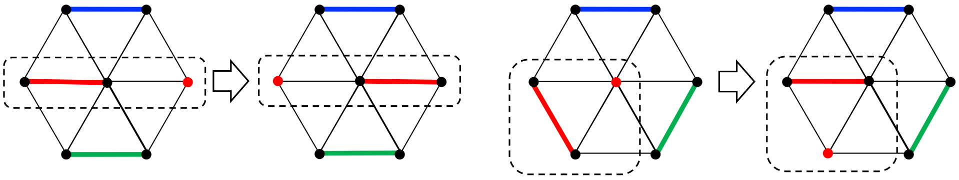


Figure 3 Slide operations. The colored, thick edges correspond to pieces.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with $2n + 1$ vertices. For a vertex u, we denote by $N(u)$ the set of vertices adjacent to u. For a vertex subset X, the subgraph induced by X in G is denoted by $G[X]$. A path or a cycle is *odd* if it has an odd number of edges.

A *matching* is a subset of edges that have no common end vertices. A matching is *nearly perfect* if its size is n. A vertex is *covered by* a matching M if it is the end vertex of some edge in M, and *exposed by* M if it is not covered by M. A cycle is M*-alternating* if edges in M and $E \setminus M$ appear alternatively along C, except for one vertex (when the cycle is odd).

Reconfiguration of Labeled Matching in Triangular Grid Graphs

Consider the 2-dimensional triangular lattice of infinite size. A *triangular grid* graph is a subgraph induced by a finite number of points in the triangular lattice. See Figure 2 for example. In this paper, we also assume that a triangular grid graph is always connected. A *hole* of a triangular grid graph is a face of the graph whose boundary is a cycle of length at least 6.

We here formally define our reconfiguration problem. Let $G = (V, E)$ be a triangular grid graph with $2n + 1$ vertices. We denote $V = [2n + 1]$, where, for a positive integer x, we write $[x] = \{1, 2, \dots, x\}$.

A *placement* is a mapping $p : [n] \to E$ such that $p(i)$ and $p(j)$ have no common end vertices for every distinct i, j. We call each $p(i)$ a *piece*. Then $\{p(i) \mid i \in [n]\} \subseteq E$ forms a nearly perfect matching in G, which is denoted by M_p. Let v_p be the unique vertex exposed by M_p. We also say that a mapping p *exposes* v_p.

We define the following operations on a placement, which we call slide (see Figure 3). Suppose that there exists an integer $j \in [n]$ such that $p(j) = (u, v)$ and $(v, v_p) \in E$. Then we transform p to a placement p_{s} defined as

$$p_{\mathrm{s}}(i) = \begin{cases} p(i) & (i \neq j) \\ (v, v_p) & (i = j). \end{cases}$$

The obtained placement p_{s} exposes the vertex u. In this case, we write $p \rightsquigarrow p_{\mathrm{s}}$.

Let p, q be two distinct placements. If there exists a sequence of placements $\mu_0, \mu_1, \dots, \mu_\ell$ such that (1) $\mu_0 = p$, $\mu_\ell = q$, (2) $\mu_t \rightsquigarrow \mu_{t+1}$ for every integer $t \in \{0, 1, \dots, \ell - 1\}$, then we say that p is *reconfigured* to q. A graph is *reconfigurable* if any two placements can be reconfigured to each other.

We remark that, in the Gourds puzzle, a piece has a pair of labels (numbers), meaning that each piece has an orientation. That is, a mapping is defined from $[n]$ to $\{(u, v), (v, u) \mid (u, v) \in E\}$. This requires us to define another operation to change the orientation of pieces. Specifically, when a piece with the exposed vertex induces a triangle, we are allowed to change the orientation of the piece. Our problem does not distinguish (u, v) and (v, u), and

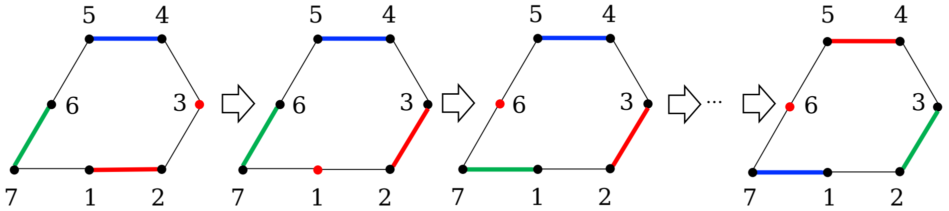


Figure 4 Rotation operations for a placement aligned with an odd cycle when $k = 3$.

a placement is defined on a mapping from $[n]$ to E. It should be noted, however, that our results can be adapted to the Gourds puzzle case with orientation. See Sections 3.1 and 4 for the details.

Rotation along a Cycle

We define a sequence of slide operations, called *rotation*, which will be used in the subsequent sections. Let C be an odd cycle. We say that a placement p is *aligned with* C if C is an odd M_p-alternating cycle and C has the exposed vertex v_p.

Let p be a placement aligned with C. In what follows, we assume for simplicity that $V(C) = [2k+1]$ for some integer $k \in [n]$, and that the vertices of C are aligned in the anti-clockwise order along C. We also assume that the first k pieces $p(1), p(2), \ldots, p(k)$ of p are located on the cycle C.

For an odd integer $j \in [2k+1]$, we define a placement p_j as, for $i \in [k]$,

$$p_j(i) = \begin{cases} (2i-1, 2i) & (2i < j) \\ (2i, 2i+1) & (j < 2i), \end{cases}$$

and $p_j(i) = p(i)$ for $i \geq k+1$. Thus p_j exposes the vertex j. Moreover, for two integers $h, j \in [2k+1]$ such that $h \equiv j \pmod 2$, define $p_{j,h}$ as, for $i \in [k]$,

$$p_{j,h}(i) = \begin{cases} (h+2i-2, h+2i-1) & (h+2i-1 < j) \\ (h+2i-1, h+2i) & (j < h+2i-1), \end{cases}$$

where these vertex labels are defined on $\mathbb{Z}_{2k+1}$ (i.e., modulo $2k+1$), and $p_{j,h}(i) = p(i)$ for $i \geq k+1$.

Figure 4 is an example when $k = 3$. The left-most figure depicts a placement $p_{3,1} = p_3$ where $(p_{3,1}(1), p_{3,1}(2), p_{3,1}(3)) = ((1,2),(4,5),(6,7))$. By applying slide to $p_{3,1}$ once, we obtain $p_{1,1} = p_1$, that is, $(p_{1,1}(1), p_{1,1}(2), p_{1,1}(3)) = ((2,3),(4,5),(6,7))$. The right-most figure depicts a placement $p_{6,4}$, which is $(p_{6,4}(1), p_{6,4}(2), p_{6,4}(3)) = ((4,5),(7,1),(2,3))$.

The following observation asserts that $p_{j,h}$'s can be reconfigured to each other in $O(k^2)$ slide operations along C. Such a sequence of slide operations is called *rotation along* C, or we say that we *rotate* p *along* C.

▶ **Observation 2.1.** *For any two odd integers $j, j' \in [2k+1]$, we can reconfigure p_j to $p_{j'}$ using at most k slide operations. Moreover, for any four integers $j, j' \in [2k+1]$ and $h, h' \in [2k+1]$ such that $j \equiv h$ and $j' \equiv h' \pmod 2$, we can reconfigure $p_{j,h}$ to $p_{j',h'}$ using at most $k^2 + k$ slide operations.*

Proof. Observe that, applying slide to p_j along C, the exposed vertex j is moved to $j-2$ or $j+2 \pmod{2k+1}$. This means that $p_j \rightsquigarrow p_{j-2}$ and $p_j \rightsquigarrow p_{j+2}$ hold. Hence, by repeating slide operations at most k times, p_j can be reconfigured to $p_{j'}$. We next show the second

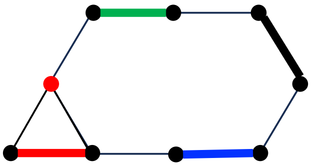


Figure 5 A factor-critical graph that is not reconfigurable. We cannot change the ordering of the pieces by slide.

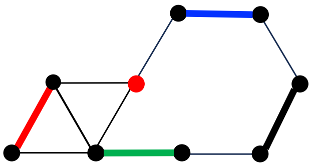


Figure 6 A factor-critical graph that is reconfigurable.

statement. Similarly to the first statement, we can reconfigure $p_{j,h}$ to $p_{h,h}$ in at most k slide operations. Applying slide to $p_{h,h}$ along C, we obtain a placement $p_{h-2,h+1}$. Repeating this procedure at most k times, we can reconfigure $p_{j,h}$ to $p_{h'-3,h'}$. Since we can reconfigure $p_{h'-3,h'}$ to $p_{j',h'}$ in at most k slide operations, the total number of slide operations is at most $k^2 + k$. ◀

3 Reconfiguration on Factor-Critical Graphs

In this section, we discuss the reconfigurability of a factor-critical graph. Recall that a graph is *factor-critical* if, for any vertex v, G has a nearly perfect matching that does not cover v.

As mentioned in Introduction, being a factor-critical graph is a necessary condition for reconfigurability.

▶ **Observation 3.1.** *If a triangular grid graph G is reconfigurable, then it is factor-critical.*

Proof. If G is not factor-critical, then G has some vertex u such that every nearly perfect matching covers u. Then we cannot move the piece covering u in an initial placement so that u becomes exposed. Hence there exist two placements such that one cannot be reconfigured to the other. Thus the observation holds. ◀

Moreover, as observed in Hamersma et al. [13], the 2-connectivity is necessary for a graph to be reconfigurable. We remark that, even if a graph is 2-connected and factor-critical, it may not be reconfigurable. See Figure 5.

The main theorem of this section is the following. We show that a graph is reconfigurable if it has a vertex of degree 6, which corresponds to a vertex not on the boundary cycles of the holes or the outer face.

▶ **Theorem 3.2.** *Let $G = (V, E)$ be a 2-connected factor-critical triangular grid graph. If G has a vertex of degree 6, then G is reconfigurable.*

We remark that our condition is not necessary, as there exists a 2-connected factor-critical graph such that it does not have a vertex of degree 6, but it is reconfigurable. See Figure 6 (see also Lemma 3.14 and Section 5).

3.1 Proof Overview

In this section, we present the proof overview of Theorem 3.2. The proof makes use of the ear decomposition of a factor-critical graph to design a reconfiguration sequence.

An *ear decomposition* of a graph G is a sequence of subgraphs $G_1, G_2, \ldots, G_k = G$ starting from a subgraph G_1 such that G_{i+1} is obtained from G_i by adding an ear P_i for each $i \geq 1$, where an *ear* P of a subgraph G' is a path of G with end vertices in G' such that P is

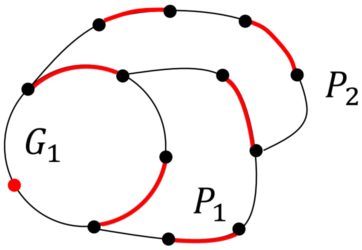


Figure 7 An ear decomposition and a matching aligned with the ear decomposition.

internally disjoint from G'. We denote by $G' + P$ the subgraph obtained from G' by adding the ear P. Thus, in the ear decomposition, it holds that $G_{i+1} = G_i + P_i$ for each $i \in [k-1]$. See Figure 7 for an example.

An ear decomposition is *proper* if the end vertices of each ear are distinct, and *odd* if each ear is of odd length. It is known that a 2-connected factor-critical graph is characterized by odd and proper ear decomposition.

▶ **Proposition 3.3** (Theorem 5.5.2 in Lovász–Plummer [21]). *A graph G is 2-connected and factor-critical if and only if G has an odd and proper ear decomposition starting from an odd cycle.*

Let p be a placement of G. Recall that M_p denotes a nearly perfect matching $\{p(i) \in E \mid i \in [n]\}$, and v_p is the vertex exposed by M_p. We say that a placement p is *aligned with an ear decomposition* $G_1, \ldots, G_k$ if it satisfies the following two conditions (Figure 7).

(a) G_1 is an odd M_p-alternating cycle with the exposed vertex v_p.

(b) For each $i \in [k-1]$, the ear P_i is M_p-alternating and its end vertices are not covered by $M_p \cap E(P_i)$.

We show that any placement p can be reconfigured so that the obtained placement p' is aligned with a given ear decomposition $G_1, \ldots, G_k$. See Section 3.2 for the proof.

▶ **Lemma 3.4.** *Let G be a 2-connected factor-critical triangular grid graph with $2n+1$ vertices. Let $G_1, \ldots, G_k$ be an odd and proper ear decomposition of G. Then we can reconfigure any placement p to a placement aligned with $G_1, \ldots, G_k$ in $O(kn)$ slide operations.*

Let q be a target placement of G. Applying Lemma 3.4 to q as well, we can reconfigure q so that the obtained placement q' is aligned with $G_1, \ldots, G_k$. By taking the inverse of the reconfiguration steps, we see that q' can be reconfigured to q in $O(kn)$ slide operations. Therefore, in order to reconfigure p to q, it suffices to reconfigure p' to q'.

We now present how to find a reconfiguration sequence between two placements aligned with $G_1, \ldots, G_k$. Since G_i is factor-critical for any $i \in [k]$, the ear structure suggests to design a reconfiguration sequence recursively. In fact, we will show in Lemma 3.9 (Section 3.3) that, if G_j is a reconfigurable graph with at least 5 vertices for some $j < k$, then so is $G_k = G$. Note that the lemma holds even if a graph is not a triangular grid graph. However, G_j's may not necessarily be reconfigurable, as there exists a factor-critical graph which is not reconfigurable. To overcome the difficulty, we introduce a special kind of ear decomposition starting from simple reconfigurable subgraphs.

We say that an odd and proper ear decomposition $G_1, G_2, \ldots, G_k$ is *admissible* if it satisfies either

(i) G_1 is a cycle of length 5 (Figure 8), or

(ii) P_1 is of length 3 and has the end vertices u, v which are adjacent in G_1 (Figure 9).

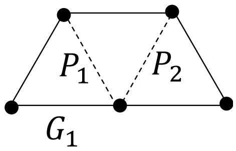


Figure 8 A pentagon.

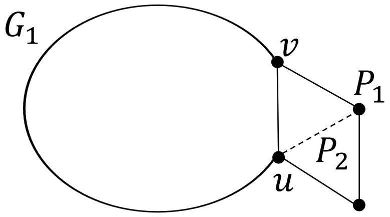


Figure 9 An odd cycle with a diamond.

Consider the case when (i) is satisfied. Since the ordering of ears with length 1 may be changed in the ear decomposition, we may assume that the first 2 ears P_1 and P_2 are the inner edges of G_1. Then $G_3 = G_1 + P_1 + P_2$ induces a pentagon in G, where a *pentagon* is a subgraph induced by three adjacent triangles. It is not difficult to see that the pentagon is reconfigurable in a constant number of slide operations. On the other hand, consider the case when (ii) is satisfied. Similarly to the case (i), we may assume that the second ear P_2 is the inner edge of P_1. The subgraph $G_3 = G_1 + P_1 + P_2$ induces an odd cycle attached to a diamond, where a *diamond* is a subgraph induced by two adjacent triangles. The subgraph is shown to be reconfigurable in Lemma 3.14.

Therefore, G_3 in an admissible ear decomposition is reconfigurable. Hence, if there exists an admissible ear decomposition, we can find a reconfiguration sequence as below. See Section 3.3 for the details.

▶ **Lemma 3.5.** *Let $G_1, \ldots, G_k = G$ be an admissible ear decomposition of G. Then any two placements aligned with the ear decomposition can be reconfigured to each other.*

We remark that the length of a reconfiguration sequence obtained in the above lemma is at most n^{2n} for a graph with $2n + 1$ vertices, which is not bounded by a polynomial in n. It may be interesting to find the optimal bound on the length of reconfiguration sequences. It is known in [13] that the length is $\Omega(n^2)$.

Finally, we show that there always exists an admissible ear decomposition in a triangular grid graph with a vertex of degree 6.

▶ **Theorem 3.6** ($\star$). *Let G be a 2-connected factor-critical triangular grid graph such that it has a vertex of degree 6. Then G has an admissible ear decomposition.*

Theorem 3.6 is a graph-theoretical result independent of designing a reconfiguration sequence. Theorem 3.6 can be proved by investigating the matching structure of factor-critical triangular grid graphs.

In summary, a reconfiguration sequence from an initial placement p to a target placement q can be realized as below, which completes the proof of Theorem 3.2.

1. Reconfigure p to a placement aligned with an admissible ear decomposition $G_1, \ldots, G_k$, denoted by p', by Lemma 3.4.
2. Using Lemma 3.5, reconfigure p' to another placement q' aligned with $G_1, \ldots, G_k$, where q' is a placement obtained from q by Lemma 3.4.
3. Reconfigure q' to the target placement q.

We remark that the proof of Theorem 3.2 above can be adapted to the Gourds puzzle in which a piece has an orientation. This is because the structures (i) and (ii) in an admissible ear decomposition can also be used to change the orientation of pieces in an arbitrary way. Thus we have the following corollary.

▶ **Corollary 3.7.** *Let B be a hexagonal grid such that the dual triangular grid graph is a 2-connected factor-critical graph with a vertex of degree 6. Then any two configurations of the same set of pieces on B can be reconfigured to each other.*

3.2 Reconfiguration to a Placement Aligned with Ear Decomposition

In this subsection, we will show Lemma 3.4, that is, we will show that we can reconfigure an initial placement p to a placement aligned with a given odd and proper ear decomposition $G_1, \dots, G_k$.

We first prove that we can reconfigure so that any vertex is exposed.

▶ **Lemma 3.8.** *Let G be a 2-connected factor-critical triangular grid graph with $2n + 1$ vertices. For any vertex v, we can reconfigure a placement p so that v is the exposed vertex, in $O(n)$ slide operations.*

Proof. Since G is factor-critical, G has a nearly perfect matching M_v that exposes v. The symmetric difference $M_p \Delta M_v$ contains an M_p-alternating path P from v_p to v, which is of even length. We reconfigure p by sliding the pieces on the path P one-by-one. The resulting placement exposes v. The number of slide operations is $|P|/2$, which is $O(n)$. ◀

To obtain a placement aligned with the ear decomposition, we first reconfigure so that an end vertex of the last ear P_{k-1} is exposed using Lemma 3.8. Then, since each inner vertex of the last ear P_{k-1} is of degree 2 in G, the obtained placement is aligned with P_{k-1}. By applying this procedure repeatedly for each ear, we can obtain a placement aligned with $G_1, \dots, G_k$. This implies Lemma 3.4 as below.

Proof of Lemma 3.4. Let v_i be an end vertex of ear P_i for $i \in [k-1]$. The basic observation is that, each inner vertex of the last ear P_{k-1} is of degree 2, and hence, if a nearly perfect matching M exposes v_{k-1}, the last ear P_{k-1} is an M-alternating path such that the end vertices are not covered by edges of $M \cap E(P_{k-1})$.

We perform the following procedure for each $i = k-1, k-2, \dots, 1$. Note that G_i is factor-critical for any $i \in [k]$.

1. Applying Lemma 3.8 to G_{i+1}, we reconfigure the current placement of G_{i+1} so that v_i is exposed. Then the resulting placement is aligned with P_i by the above observation.

In the end of the above procedure, the obtained placement of the original graph G is aligned with $P_{k-1}, \dots, P_1$. Moreover, the exposed vertex is on G_1. Thus this is a desired placement. The necessary number of slide operations is $O(kn)$, since we repeat the procedure of Lemma 3.8 $k-1$ times. ◀

3.3 Reconfiguration using Ear Decomposition

We next present how to reconfigure a placement aligned with an ear decomposition. Using the ear structure, we can find a reconfiguration sequence if the subgraph G_{k-1} is reconfigurable.

▶ **Lemma 3.9.** *Let $G_1, \dots, G_k$ be an odd and proper ear decomposition of a graph G with $2n + 1$ vertices. Suppose that G_{k-1} has at least 5 vertices, and that, in G_{k-1}, any placement aligned with the ear decomposition $G_1, \dots, G_{k-1}$ can be reconfigured to another placement aligned with $G_1, \dots, G_{k-1}$, using t slide operations. Then there exists a reconfiguration sequence between any two placements along with $G_1, \dots, G_k$ in G, which requires $O(n^2(t+n))$ slide operations.*

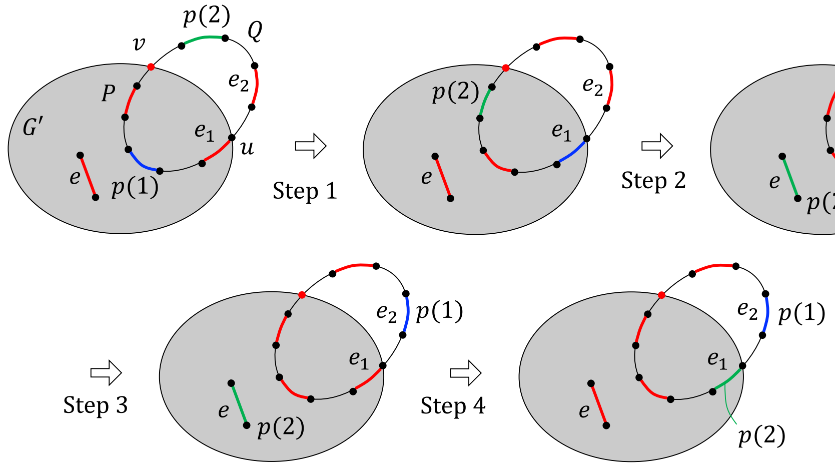


Figure 10 Proof of Claim 3.11: An example when $j = 2$.

Proof. For simplicity, we denote $G_{k-1} = G'$ and $P_{k-1} = Q$ in the proof. Let u, v be the end vertices of Q. Let p be an initial placement of G and q be a target placement of G, both of which are aligned with the ear decomposition. It follows from Lemma 3.8 that we can reconfigure p (and q, resp.) so that v is exposed. Hence we may assume that both p and q expose v. Moreover, by changing the indices of the pieces if necessary, we may assume that the pieces $q(1), \ldots, g(\ell)$ are placed on the ear Q in the order from v to u.

▷ Claim 3.10. There exists an M_p-alternating path P of even length from v to u in G'.

Proof. Since G' is factor-critical, there exists a nearly perfect matching M_u that exposes u. Taking $M_p \triangle M_u$, we see that there exists an even M_p-alternating path P from v to u in G'. ◁

Let C be the cycle consisting of P and Q. Then C is an odd M-alternating cycle in G. We will show that we can reconfigure p so that the first ℓ pieces are on the ear Q, using the cycle C. We consider the following two cases, depending on whether C is a Hamilton cycle or not.

▷ Claim 3.11. Suppose that C is not a Hamilton cycle of G. Then we can reconfigure p to a placement p' so that $p'(1), \ldots, p'(\ell)$ are placed on Q in this order from v to u, using $O(\ell(t + n^2))$ slide operations.

Proof. In this case, the graph G' has an edge $e \notin E(P)$ such that $e \in M_p$. Let e_1 be the edge of M_p covering u, and e_2 be the edge of of M_p covering the vertex adjacent to u on Q. See Figure 10.

We may assume that $p(1)$ is on the cycle C, as otherwise $p(1)$ is contained in G', and hence we can reconfigure the current placement in G' so that $p(1)$ is on P, keeping v exposed, using t slide operations by the assumption.

We reconfigure p by the following 4 steps for $j = 2, \ldots, \ell$. Initially, we set $\tilde{p} = p$.

1. We move the piece $\tilde{p}(j)$ so that $\tilde{p}(j)$ is on P as follows.
   a. If $\tilde{p}(j)$ is on Q, we rotate the current placement $\tilde{p}$ along C so that $\tilde{p}(j)$ is located on P.
   b. If $\tilde{p}(j)$ is contained in G' but not on P, then we reconfigure the current placement $\tilde{p}$ on G' so that $\tilde{p}(j)$ is located on P, keeping that $\tilde{p}(1), \ldots, \tilde{p}(j-1)$ are on C.

2. We reconfigure the current placement $\tilde{p}$ on G' to swap $\tilde{p}(j)$ and the piece on e. Thus $\tilde{p}(j) = e$.
3. We rotate the current placement $\tilde{p}$ along C so that $\tilde{p}(j-1)$ is e_2.
4. We reconfigure the current placement $\tilde{p}$ on G' to swap $\tilde{p}(j)$ and the piece on e_1. Thus $\tilde{p}(j)$ has been changed to e_1.

In the end of the j-th iteration, $\tilde{p}(1), \ldots, \tilde{p}(j)$ are located on C in this order from v to u. Therefore, in the end of the above procedure, the pieces $\tilde{p}(1), \ldots, \tilde{p}(\ell)$ are located on C in this order from v to u. Thus we can rotate $\tilde{p}$ along C so that they are on Q.

In the above procedure, for each j, we reconfigure the placement restricted on G' in a constant number of times, and we rotate the placement along C at most twice. Therefore, the total number of slide operations is $O(\ell(t+n^2))$ by Observation 2.1. ◁

▷ Claim 3.12. Suppose that C is a Hamilton cycle of G. Then we can reconfigure p to a placement p' so that $p'(1), \ldots, p'(\ell)$ are placed on Q in this order from v to u, using $O(\ell n(t+n))$ slide operations.

Proof. Since G' has at least 5 vertices, P has at least 2 edges of M_p. Let e_1, e_2 be two edges in $M_p \cap E(P)$ such that e_1, e_2 appear consecutively along P. We can swap the 2 pieces on e_1 and e_2 by reconfiguring on G', using t slide operations. By using the strategy similar to the bubble sort algorithm, we can obtain a placement p' such that $p'(1), \ldots, p'(\ell)$ are on C in this order from v to u. This requires $O(\ell n)$ swaps. Since each swap takes $O(t+n)$ slide operations, it takes $O(\ell n(t+n))$ slide operations in total. ◁

In each case, we can reconfigure p to a placement p' so that the pieces $p'(1), \ldots, p'(\ell)$ are located on Q in this order from v to u. Since we can reconfigure the placement on G' to any placement, we can reconfigure p' to q. The total number of slide operations is $O(\ell n(t+n)) = O(n^2(t+n))$. ◀

By applying Lemma 3.9 recursively, we see that, if G_j is a reconfigurable subgraph with at least 5 vertices for some $j < k$, then $G_k = G$ is reconfigurable. In particular, if a given ear decomposition is admissible, then G is shown to be reconfigurable.

Below we upper-bound the number of operations to reconfigure two placements aligned with an admissible ear decomposition. We will show each case of the definition of an admissible ear decomposition separately. We assume that a graph has $2n+1$ vertices for $n \geq 2$.

▶ **Lemma 3.13** (⋆). *Let $G_1, \ldots, G_k$ be an admissible ear decomposition such that G_1 is a cycle of length 5 (Figure 8). Then we can reconfigure an arbitrary placement p aligned with $G_1, \ldots, G_k$ to another placement aligned with $G_1, \ldots, G_k$ in at most n^{2n} slide operations.*

We next discuss the second case of an admissible ear decomposition. The following lemma says that the base case is reconfigurable.

Let $\tilde{G} = (V, E)$ be a triangular grid graph with $2n+1$ vertices for $n \geq 3$ as in Figure 11. More specifically, $V = [2n+1]$, and it consists of an odd cycle C of length $2n-1$ with vertex set $[2n-1]$, attached to a diamond D with vertex set $\{2n-2, 2n-1, 2n, 2n+1\}$.

▶ **Lemma 3.14.** *The graph $\tilde{G}$ defined above with $2n+1$ vertices ($n \geq 3$) is reconfigurable in at most $n^3 + n^2$ operations.*

Proof. Let p and q be an initial and target placements of $\tilde{G}$, respectively. We may assume that the target pieces $q(1), \ldots, q(n-1)$ are located in the anti-clockwise order along C, and that D has pieces $q(n-1)$ and $q(n)$. Let C' be the Hamilton cycle of length $2n+1$ in $\tilde{G}$.

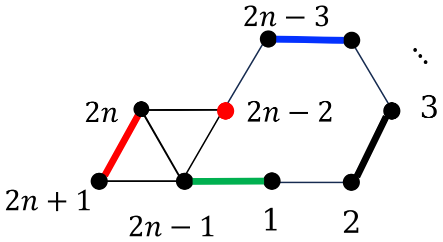


Figure 11 A factor-critical graph that is reconfigurable.

We present a reconfiguration sequence as follows: Initially, we set $\tilde{p} = p$. For $j = 1, 2, \ldots, n$, we do the following 2 steps.

1. We rotate the current placement $\tilde{p}$ along C' so that $\tilde{p}(j)$ is equal to the edge $(2n, 2n+1)$.
2. We rotate the current placement $\tilde{p}$ along C so that $\tilde{p}(j-1)$ is equal to the edge $(2n-1, 1)$. Then $\tilde{p}(1), \ldots, \tilde{p}(j)$ are located on C' in the anti-clockwise order.

In the end of the procedure, $\tilde{p}(1), \ldots, \tilde{p}(n)$ are located on C' in the anti-clockwise order, which is the desired placement q. In each iteration, we rotate the current placement along C or C'. By choosing the shorter one between the clockwise rotation and the anti-clockwise rotation, it requires at most $n^2 + n$ slide operations by Observation 2.1. Hence the total number of slide operations is at most $n^3 + n^2$. ◀

We next show the case when an admissible ear decomposition satisfies the second case. This, together with Lemma 3.13, proves Lemma 3.5.

▶ **Lemma 3.15** (⋆). *Let $G_1, \ldots, G_k$ be an admissible ear decomposition such that P_1 is of length 3 and has the end vertices u, v which are adjacent in G_1 (Figures 9 and 11). Then we can reconfigure an arbitrary placement p aligned with $G_1, \ldots, G_k$ to another placement in at most n^{2n} slide operations.*

4 Reconfiguration on Locally-Connected Graphs

In this section, we consider a triangular grid graph which is locally-connected. A vertex u in a graph G is said to be *locally-connected* if the subgraph $G[N(u)]$ is connected. A graph G is called *locally-connected* if every vertex is locally-connected. It is observed that, if G is locally-connected, then it is 2-connected, since, for a cut vertex u, the subgraph $G[N(u)]$ is disconnected.

The following theorem says that a locally-connected triangular grid graph, except for the Star of David graph (Figure 1), is Hamiltonian. We note that, since their proof is constructive, a Hamilton cycle can be found in polynomial time.

▶ **Theorem 4.1** (Gordon, Orlovich, and Werner [12]). *Let G be a triangular grid graph. If G is locally-connected, but not isomorphic to the Star of David graph, then it has a Hamilton cycle.*

It follows from the above theorem that a locally-connected triangular grid graph, except for the Star of David graph, is factor-critical, as it has an odd and proper ear decomposition starting from a Hamilton cycle such that all the ears are single edges. On the other hand, the Star of David graph is not factor-critical, and hence it is not reconfigurable (see also [13]).

The main theorem of this section is the following.

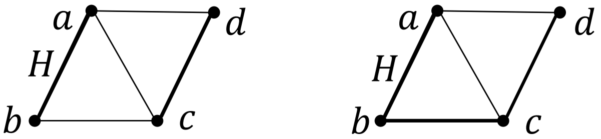


Figure 12 Diamonds and Hamilton cycles satisfying the condition in Lemma 4.5.

▶ **Theorem 4.2.** *Let $G = (V, E)$ be a triangular grid graph with $2n + 1$ vertices. If G is locally-connected, but not isomorphic to the Star of David graph, then G is reconfigurable. Moreover, a reconfiguration sequence using $O(n^3)$ slide operations can be found in polynomial time.*

The proof for Theorem 4.2 exploits a Hamilton cycle in G to design a reconfiguration sequence. We note that the proof for 2-connected graphs with no holes by Hamersma et al. [13] also uses a Hamilton cycle. Our proof refines their proof so that we can deal with holes.

Suppose that we are given two placements p and q. The proposed algorithm to reconfigure p to q consists of the following three phases.

1. Reconfigure p to a placement aligned with a Hamilton cycle H, denoted by p'.
2. Reconfigure p' to another placement q' aligned with H.
3. Reconfigure q' to the target placement q.

In Phase 1, we first reconfigure the initial placement p to a placement aligned with the Hamilton cycle H, which is denoted by p'. Applying the same procedure to the target placement q, we obtain a placement aligned with H, denoted by q'. We then reconfigure p' to q' in Phase 2. In Phase 3, the placement q' can be reconfigured to the target placement q by taking the inverse of Phase 1 operations for q.

It was shown in Hamersma et al. [13] that we can reconfigure a placement to a placement aligned with a Hamilton cycle. We recall that a locally-connected graph is 2-connected.

▶ **Theorem 4.3** (Hamersma et al. [13])**.** *Let G be a 2-connected triangular grid graph with $2n + 1$ vertices. Then we can reconfigure any placement to a placement aligned with a Hamilton cycle H, using $O(n^2)$ slide operations.*

Therefore, Phases 1 and 3 can be implemented in $O(n^2)$ slide operations. Thus it suffices to implement Phase 2 to reconfigure any placement aligned with H to another placement aligned with H. This step is realized by the following theorem.

▶ **Theorem 4.4.** *Let G be a triangular grid graph with $2n + 1$ vertices, which is locally-connected, but not isomorphic to the Star of David graph. Let H be a Hamilton cycle. For a pair of placements p, q aligned with H, we can reconfigure p to q in $O(n^3)$ slide operations.*

The proof of Theorem 4.4 adopts a similar strategy to that of Theorem 3.2, where we employs a Hamilton cycle instead of an ear decomposition. We identify a small subgraph that can be used to reconfigure placements aligned with H.

▶ **Lemma 4.5** ($\star$)**.** *Let G be a triangular grid graph with $2n + 1$ vertices. Let H be a Hamilton cycle of G. Suppose that G has a diamond, whose vertices are a, b, c, d aligned in the anti-clockwise order (Figure 12), such that either*

(i) *H contains the edges (a, b) and (c, d), but does not contain (a, c), or*

(ii) *H contains the edges (a, b) and (b, c).*

Then we can reconfigure any placement aligned with H to another placement aligned with H in $O(n^3)$ slide operations.

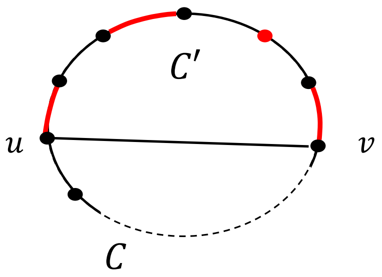


Figure 13 An odd cycle with one chord.

We then show that such a diamond with a Hamilton cycle always exists if G is a locally-connected triangular grid graph, which is not isomorphic to the Star of David graph. This shows Theorem 4.4.

▶ **Lemma 4.6** ($\star$). *Let G be a triangular grid graph with $2n+1$ vertices. If G is a locally-connected triangular grid graph, which is not isomorphic to the Star of David graph, then there exist a Hamilton cycle H and a diamond, whose vertices are denoted by a, b, c, d (Figure 12), that satisfy either* (i) *or* (ii) *in Lemma* 4.5*.*

This section is concluded with stating our results on the Gourds puzzle.

▶ **Corollary 4.7.** *Let B be a hexagonal grid such that the dual graph is locally-connected, but not isomorphic to the Star of David graph. Then any two configurations of the same set of n pieces on B can be reconfigured to each other, using $O(n^3)$ moves.*

5 Concluding Remarks

In this paper, we introduced a new reconfiguration problem of labeled matchings in a triangular grid graph. We provided sufficient conditions for a graph to be reconfigurable using a factor-critical graphs and a locally-connected graphs. It remains open to characterize a reconfigurable triangular grid graph, when it is a factor-critical graph with no vertex of degree 6, but not locally-connected. Let us here discuss the difficulty to obtain the characterization. For example, consider a graph G consisting of an odd cycle C of length $2n+1$ with one chord (u, v) (Figure 13). Let C' be the odd cycle of G with the edge (u, v). The length of C' is denoted by $2m+1$. Then we can observe that the reconfigurability of G depends on m and n. Specifically, G is reconfigurable if and only if $n-1$ and $m-1$ are mutually prime. Indeed, since we can only rotate a placement along C' and C, which correspond to cyclic permutations on $[m]$ and $[n]$, respectively, any permutation can be realized if and only if $n-1$ and $m-1$ are mutually prime. This observation would imply that it requires algebraic conditions to characterize a reconfigurable graph, like the 15-puzzle.

References

1 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.3.

2 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *J. Comput. Syst. Sci.*, 136:220–248, 2023. doi:10.1016/J.JCSS.2023.03.008.

3 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 80:1–80:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.MFCS.2019.80.

4 Édouard Bonnet, Tillmann Miltzow, and Pawel Rzazewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018. doi:10.1007/S00453-017-0387-0.

5 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *CoRR*, abs/2204.10526, 2022. doi:10.48550/arXiv.2204.10526.

6 Josh Brunner, Lily Chung, Erik D. Demaine, Dylan H. Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 1 X 1 rush hour with fixed blocks is pspace-complete. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FUN.2021.7.

7 Kevin Buchin and Maike Buchin. Rolling block mazes are pspace-complete. *J. Inf. Process.*, 20(3):719–722, 2012. doi:10.2197/IPSJJIP.20.719.

8 Jean Cardinal and Raphael Steiner. Inapproximability of shortest paths on perfect matching polytopes. In Alberto Del Pia and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization - 24th International Conference, IPCO 2023, Madison, WI, USA, June 21-23, 2023, Proceedings*, volume 13904 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2023. doi:10.1007/978-3-031-32726-1_6.

9 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015. doi:10.1016/J.TCS.2015.07.037.

10 Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$-puzzle is hard. *Theor. Comput. Sci.*, 732:80–84, 2018. doi:10.1016/J.TCS.2018.04.031.

11 Gary William Flake and Eric B. Baum. Rush hour is pspace-complete, or "why you should generously tip parking lot attendants". *Theor. Comput. Sci.*, 270(1-2):895–911, 2002. doi:10.1016/S0304-3975(01)00173-6.

12 Valery S. Gordon, Yury L. Orlovich, and Frank Werner. Hamiltonian properties of triangular grid graphs. *Discret. Math.*, 308(24):6166–6188, 2008. doi:10.1016/J.DISC.2007.11.040.

13 Joep Hamersma, Marc J. van Kreveld, Yushi Uno, and Tom C. van der Zanden. Gourds: A sliding-block puzzle with turning. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 33:1–33:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ISAAC.2020.33.

14 Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/J.TCS.2005.05.008.

15 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation.* A K Peters, 2009.

16 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.

17 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. *SIAM J. Discret. Math.*, 36(2):1102–1123, 2022. doi:10.1137/20M1364370.

18 Wm. Woolsey Johnson and William E. Story. Notes on the “15” puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. URL: http://www.jstor.org/stable/2369492.

19 Naonori Kakimura and Yuta Mishima. Reconfiguration of labeled matchings in triangular grid graphs, 2024. arXiv:2409.11723.

20 Dohan Kim. Sorting on graphs by adjacent swaps using permutation groups. *Comput. Sci. Rev.*, 22:89–105, 2016. doi:10.1016/J.COSREV.2016.09.003.

21 L. Lovász and M.D. Plummer. *Matching Theory.* AMS Chelsea Publishing Series. AMS Chelsea Pub., 2009.

22 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 66:1–66:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ESA.2016.66.

23 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

24 Daniel Ratner and Manfred K. Warmuth. Finding a shortest solution for the N × N extension of the 15-puzzle is intractable. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 168–172. Morgan Kaufmann, 1986. URL: http://www.aaai.org/Library/AAAI/1986/aaai86-027.php.

25 Laura Sanità. The diameter of the fractional matching polytope and its hardness implications. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 910–921, 2018. doi:10.1109/FOCS.2018.00090.

26 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, Berlin, 2003.

27 J. Slocum and D. Sonneveld. *The 15 Puzzle Book: How It Drove the World Crazy.* Slocum Puzzle Foundation, 2006.

28 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

Composition Orderings for Linear Functions and Matrix Multiplication Orderings

Susumu Kubo
General Education Center, Tottori University of Environmental Studies, Japan

Kazuhisa Makino
Research Institute for Mathematical Sciences, Kyoto University, Japan

Souta Sakamoto
Acompany Co., Ltd., Nagoya, Japan

Abstract

We first consider composition orderings for linear functions of one variable. Given n linear functions $f_1, \dots, f_n : \mathbb{R} \to \mathbb{R}$ and a constant $c \in \mathbb{R}$, the objective is to find a permutation $\sigma : [n] \to [n]$ that minimizes/maximizes $f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}(c)$, where $[n] = \{1, \dots, n\}$. It was first studied in the area of time-dependent scheduling, and known to be solvable in $O(n \log n)$ time if all functions are nondecreasing. In this paper, we present a complete characterization of optimal composition orderings for this case, by regarding linear functions as two-dimensional vectors. We also show the equivalence between local and global optimality in optimal composition orderings. Furthermore, by using the characterization above, we provide a fixed-parameter tractable (FPT) algorithm for the composition ordering problem with general linear functions, with respect to the number of decreasing linear functions.

We next deal with matrix multiplication as a generalization of composition of linear functions. Given n matrices $M_1, \dots, M_n \in \mathbb{R}^{m \times m}$ and two vectors $\boldsymbol{w}, \boldsymbol{y} \in \mathbb{R}^m$, where m is a positive integer, the objective is to find a permutation $\sigma : [n] \to [n]$ that minimizes/maximizes $\boldsymbol{w}^\top M_{\sigma(n)} \cdots M_{\sigma(1)} \boldsymbol{y}$. The matrix multiplication ordering problem has been studied in the context of max-plus algebra, but despite being a natural problem, it has not been explored in the conventional algebra to date. By extending the results for composition orderings for linear functions, we show that the matrix multiplication ordering problem with 2×2 matrices is solvable in $O(n \log n)$ time if all the matrices are simultaneously triangularizable and have nonnegative determinants, and FPT with respect to the number of matrices with negative determinants, if all the matrices are simultaneously triangularizable. As the negative side, we prove that three possible natural generalizations are NP-hard. In addition, we derive the existing result for the minimum matrix multiplication ordering problem with 2×2 upper triangular matrices in max-plus algebra, which is an extension of the well-known Johnson's rule for the two-machine flow shop scheduling, as a corollary of our result in the conventional algebra.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases function composition, matrix multiplication, ordering problem, scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.44

Related Version *Full Version*: `https://arxiv.org/abs/2402.10451` [18]

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society."
Susumu Kubo: Partially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Leading Initiative for Excellent Young Researchers Grant Number JPMXS0320200347.
Kazuhisa Makino: Partially supported by JSPS KAKENHI Grant Numbers JP20H05967 and JP19K22841.

Acknowledgements The authors thank Kristóf Bérczi (Eötvös Loránd Univ.), Yasushi Kawase (Univ. of Tokyo), and Takeshi Tokuyama (Kwansei Gakuin Univ.) for their helpful comments on the first version of this paper.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 44; pp. 44:1–44:14
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We first consider composition orderings for linear functions, that is, polynomial functions of degree one or zero. Namely, given a constant $c \in \mathbb{R}$ and n linear functions $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}$, each of which is expressed as $f_i(x) = a_i x + b_i$ for some $a_i, b_i \in \mathbb{R}$, we find a permutation $\sigma : [n] \to [n]$ that minimizes/maximizes $f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}(c)$, where $[n] = \{1, \ldots, n\}$ for a positive integer n. Since composition of functions is *not* commutative even for linear functions, i.e., $f_{\sigma(2)} \circ f_{\sigma(1)} \neq f_{\sigma(1)} \circ f_{\sigma(2)}$ holds in general, it makes sense to investigate the problem. For example, let $f_1(x) = -(1/2)x + 3/2$, $f_2(x) = x - 3$, $f_3(x) = 3x - 1$, and $c = 0$, then the identity σ (i.e., $\sigma(1) = 1$, $\sigma(2) = 2$ and $\sigma(3) = 3$) provides $f_3 \circ f_2 \circ f_1(0) = f_3(f_2(f_1(0))) = f_3(f_2(3/2)) = f_3(-3/2) = -11/2$, while the permutation τ with $\tau(1) = 2$, $\tau(2) = 1$ and $\tau(3) = 3$ provides $f_3 \circ f_1 \circ f_2(0) = 8$. In fact, we can see that σ and τ are respectively minimum and maximum permutations for the problem. The composition ordering problem is natural and fundamental in many fields such as combinatorial optimization, computer science, and operations research. This problem, which was introduced by Kawase, Makino and Seimi [17], had been dealt with implicitly in the field of scheduling.

The problem was first studied from an algorithmic point of view under the name of *time-dependent scheduling* (e.g., [8, 9]). We are given n jobs with processing times $p_1, \ldots, p_n$. Unlike the classical scheduling, the processing time p_i is *not* constant, depending on the starting time of job i. Here each p_i is assumed to satisfy $p_i(s) \leq t + p_i(s + t)$ for any positive reals s and t, since we should be able to finish processing job i earlier if it starts earlier. The model was introduced to deal with learning and deteriorating effects. As the most fundamental setting of the time-dependent scheduling, we consider the linear model of single-machine makespan minimization, where the makespan denotes the time when all the jobs have been processed, and we assume that the machine can handle only one job at a time and preemption is not allowed. The linear model means that the processing time p_i is linear in the starting time s, i.e., $p_i(s) = \tilde{a}_i s + \tilde{b}_i$ for some constants $\tilde{a}_i$ and $\tilde{b}_i$. Then it is not difficult to see that the model can be regarded as the minimum composition ordering problem with linear functions $f_i(x) = (\tilde{a}_i + 1)x + \tilde{b}_i$, since f_i represents the time to finish job i if the processing of the job starts at time x. Mosheiov [20] showed the makespan is independent of the schedule, i.e., any permutation provides the same composite, if $\tilde{b}_i = 0$ for any $i \in [n]$. Gawiejnowicz and Pankowska [13], Gupta and Gupta [14], Tanaev et al. [23], and Wajs [24] studied the linear deterioration model, that is, $\tilde{a}_i, \tilde{b}_i > 0$ (i.e., $a_i > 1$ and $b_i > 0$) for any $i \in [n]$. Here $\tilde{a}_i$ and $\tilde{b}_i$ are respectively called the *deterioration rate* and the *basic processing time* of job i. It can be shown that a minimum permutation can be obtained by arranging the jobs nonincreasingly with respect to $\tilde{a}_i/\tilde{b}_i \, (= (a_i - 1)/b_i)$. Gawiejnowicz and Pankowska [13] also considered the cases $\tilde{a}_i = 0$ or $\tilde{b}_i = 0$ for some i. Gawiejnowicz and Lin [12] dealt with the linear models with nonnegative coefficients for various criteria. On the other hand, Ho, Leung and Wei [15] considered the linear learning model, that is, $0 > \tilde{a}_i > -1, \tilde{b}_i > 0$ (i.e., $1 > a_i > 0$ and $b_i > 0$) for any $i \in [n]$ and showed that a minimum permutation can be obtained again by arranging the jobs nonincreasingly with respect to $\tilde{a}_i/\tilde{b}_i \, (= (a_i - 1)/b_i)$. Gawiejnowicz, Kurc and Pankowska [11] discussed the relations between the deterioration model and the learning model. Later, Kawase et al. [17] introduced the composition ordering problem, showed that the maximization problem can be formulated as the minimization one, and proposed an $O(n \log n)$-time algorithm if all f_i's are nondecreasing, i.e., $a_i \geq 0$ for any $i \in [n]$. However, it is still open whether it is polynomially computable for general linear functions. Moreover, it is not known even when constantly many functions are decreasing.

We remark that the time-dependent scheduling with the ready time and the deadline can be regarded as the composition ordering problem with piecewise linear functions, and is known to be NP-hard, and Kawase et al. [17] also studied the composition ordering for non-linear functions as well as the related problems such as partial composition and k-composition. We also remark that the free-order secretary problem, which is closely related to a branch of the problems such as the full-information secretary problem [6], knapsack and matroid secretary problems [1, 2, 22] and stochastic knapsack problems [4, 5], can also be regarded as the composition ordering problem [17].

Main results obtained in this paper

We first characterize the minimum composition orderings for increasing linear functions. In order to describe our result, we need to define three important concepts: counterclockwiseness, collinearity, and potential identity.

We view a linear function $f(x) = ax + b$ as the vector $\begin{pmatrix} b \\ 1-a \end{pmatrix}$ in $\mathbb{R}^2$, and its *angle*, denoted by $\theta(f)$, is defined as the polar angle in $[0, 2\pi)$ of the vector, where we define $\theta(f) = \perp$ if the vector of f is the origin $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, i.e., f is the identity function. For linear functions $f_1, \ldots, f_n$, a permutation $\sigma : [n] \to [n]$ is called *counterclockwise* if there exists an integer $k \in [n]$ such that $\theta(f_{\sigma(k)}) \leq \cdots \leq \theta(f_{\sigma(n)}) \leq \theta(f_{\sigma(1)}) \leq \cdots \leq \theta(f_{\sigma(k-1)})$, where identity functions f_i (i.e., $\theta(f_i) = \perp$) are ignored and the inequalities are assumed to be transitive. For example, we consider inequalities such as $\theta(f_{\sigma(1)}) \leq \theta(f_{\sigma(3)})$ if $\theta(f_{\sigma(2)}) = \perp$. Linear functions $f_1, \ldots, f_n$ are called *collinear* if the corresponding vectors lie in some line through the origin, i.e., there exists an angle λ such that $\theta(f_i) \in \{\lambda, \lambda + \pi, \perp\}$ for all $i \in [n]$, and *potentially identical* if there exists a counterclockwise permutation $\sigma : [n] \to [n]$ such that the corresponding composite is the identity function, i.e., $f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}(x) = x$. A permutation is called *minimum* (resp., *maximum*) if the corresponding composite is the minimum (resp., maximum). Then we have the following complete characterization of minimum permutations.

▶ **Theorem 1.** *For the minimum composition ordering problem with increasing linear functions $f_1, \ldots, f_n$, one of the following three statements holds.*

(i) *They are collinear if and only if any permutation is minimum.*

(ii) *If they are not collinear, then the following statements are equivalent:*

  (ii-1) *They are potentially identical.*

  (ii-2) *A permutation is minimum if and only if it is counterclockwise.*

(iii) *If they are neither collinear nor potentially identical, then a permutation σ is minimum if and only if it is a counterclockwise permutation such that $\theta(f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}) + \pi \in [\theta(f_{\sigma(t)}), \theta(f_{\sigma(s)})]_{2\pi}$, where s and t denote the first and last integers i such that $f_{\sigma(i)}$ is not the identity function.*

Here we define $[\theta_1, \theta_2]_{2\pi} = \{\theta \in [\lambda_1, \lambda_2] \mid \lambda_1 =_{2\pi} \theta_1,\ \lambda_2 =_{2\pi} \theta_2,\ \lambda_2 - \lambda_1 \in [0, 2\pi)\}$, where for two angles $\theta_1, \theta_2 \in \mathbb{R}$, we write $\theta_1 =_{2\pi} \theta_2$ if they are congruent on the angle, i.e., $\theta_1 - \theta_2 \in 2\pi\mathbb{Z}$.

Although a single minimum permutation can be computed efficiently [17], the structure of the minimum permutations has not been clarified. Therefore, it has been difficult to construct an efficient algorithm for the minimum composition ordering problem in general (including decreasing linear functions). Theorem 1 provides an interesting achievement that clarifies the structure. Moreover, we can obtain the characterization of the minimum permutations for nondecreasing linear functions by extending Theorem 1.

We note that Theorem 1 can also characterize maximum permutations by replacing "counterclockwise" by "clockwise", which is obtained from a transformation between minimization and maximization. (See (2) in Section 2 and Remark 16 in Section 3). Incidentally, the lexicographical orderings which Kawase et al. [17] introduced can be interpreted as counterclockwise permutations, and they showed the existence of counterclockwise minimum permutations.

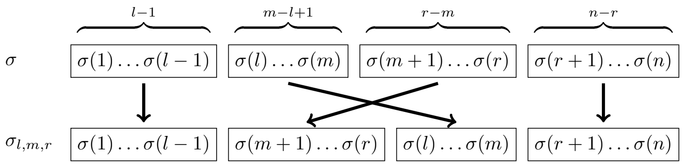
These results enable us to count and enumerate all minimum/maximum permutations efficiently.

▶ **Corollary 2.** *The number of the minimum permutations of the minimum composition ordering problem with increasing linear functions can be computed in polynomial time and there exists a polynomial delay algorithm for enumerating all of them.*

We also show the equivalence between the (global) minimality and the local minimality for increasing linear functions, which is of independent interest from an optimization point of view. To introduce the neighborhood of a permutation, let $\sigma : [n] \to [n]$ be a permutation. For three positive integers ℓ, m and r with $\ell \leq m < r$, define a permutation $\sigma_{\ell,m,r} : [n] \to [n]$ by

$$\sigma_{\ell,m,r}(i) = \begin{cases} \sigma(i) & \text{if } 1 \leq i < \ell,\ r < i \leq n, \\ \sigma(i - \ell + m + 1) & \text{if } \ell \leq i < \ell - m + r, \\ \sigma(i + m - r) & \text{if } \ell - m + r \leq i \leq r, \end{cases}$$

which is illustrated in Figure 1.



$l-1$
$m-l+1$
$r-m$
$n-r$
σ
$\sigma(1)\dots\sigma(l-1)$
$\sigma(l)\dots\sigma(m)$
$\sigma(m+1)\dots\sigma(r)$
$\sigma(r+1)\dots\sigma(n)$
$\sigma_{l,m,r}$
$\sigma(1)\dots\sigma(l-1)$
$\sigma(m+1)\dots\sigma(r)$
$\sigma(l)\dots\sigma(m)$
$\sigma(r+1)\dots\sigma(n)$

Figure 1 Permutation $\sigma_{l,m,r}$ obtained from σ by swapping two adjacent intervals.

The *neighborhood* $N(\sigma)$ of σ is defined by $N(\sigma) = \{\sigma_{\ell,m,r} \mid \ell \leq m < r\}$, that is, the set of permutations obtained from σ by swapping two adjacent intervals in σ. Note that swapping jobs and considering partial schedules (intervals) can be found in the context of a single machine time-dependent scheduling problem of minimizing the total completion time of linearly deteriorating jobs [10, 21]. A permutation σ is *locally minimum* if $f^\sigma \leq f^\mu$ for any $\mu \in N(\sigma)$, where f^σ is the composite by σ, that is, $f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}$.

▶ **Theorem 3.** *For the minimum composition ordering problem with increasing linear functions, a permutation is (globally) minimum if and only if it is locally minimum.*

The theorem reveals an interesting structural property of composition orderings. We remark that the same results hold if "minimum" is replaced by "maximum" in Corollary 2 and Theorem 3, similarly with Theorem 1. The results also hold if "increasing" is replaced by "nondecreasing".

We then deal with composition orderings for general linear functions. We provide several structural properties of minimum permutations. These, together with the characterization for increasing linear functions, provide a fixed-parameter tractable (FPT) algorithm for the minimum composition ordering problem with general linear functions, with respect to the number of decreasing linear functions.

▶ **Theorem 4.** *A minimum permutation of the minimum composition ordering problem with n linear functions can be computed in $O(k2^k n^6)$ time, where $k\,(>0)$ denotes the number of decreasing linear functions.*

We remark that the FPT algorithm can be modified to count and enumerate all minimum permutations efficiently.

We next consider the multiplication orderings for matrices as a generalization of the composition orderings for linear functions. The problem with matrices is to find a permutation $\sigma : [n] \to [n]$ that minimizes/maximizes $\boldsymbol{w}^\top M_{\sigma(n)} \cdots M_{\sigma(1)} \boldsymbol{y}$ for given n matrices $M_1, \dots, M_n \in \mathbb{R}^{m\times m}$ and two vectors $\boldsymbol{w}, \boldsymbol{y} \in \mathbb{R}^m$, where m is a positive integer. The problem has been studied in the context of max-plus algebra [3], but despite being a natural problem, to our best knowledge, it has not been explored in the conventional algebra to date.

If we set $\boldsymbol{w} = \begin{pmatrix}1\\0\end{pmatrix}$, $\boldsymbol{y} = \begin{pmatrix}0\\1\end{pmatrix}$, and $M_i = \begin{pmatrix}a_i & b_i\\ 0 & 1\end{pmatrix}$ for any $i \in [n]$, then the matrix multiplication ordering problem is (mathematically) equivalent to the composition ordering problem with linear functions $f_i(x) = a_i x + b_i$, which shows that the matrix multiplication ordering problem is a natural generalization of the composition ordering problem with linear functions.

We obtain the following generalization of the results for linear functions. Matrices $M_1, \dots, M_n \in \mathbb{R}^{m\times m}$ are called *simultaneously triangularizable* if there exists an invertible matrix $P \in \mathbb{R}^{m\times m}$ such that $P^{-1}M_iP$ is an upper triangular matrix for any $i \in [n]$.

▶ **Theorem 5.** *For the minimum matrix multiplication ordering problem with n simultaneously triangularizable 2×2 matrices, the following statements hold.*

(i) *If all matrices have nonnegative determinants, then a minimum permutation can be computed in $O(n \log n)$ time.*

(ii) *If some matrix has a negative determinant, then a minimum permutation can be computed in $O(k2^k n^6)$ time, where k denotes the number of matrices with negative determinants.*

As the negative side, we show that all possible natural generalizations turn out to be intractable unless P = NP.

▶ **Theorem 6.**

(i) *The minimum matrix multiplication ordering problem with 2×2 matrices is strongly NP-hard, even if all matrices are nonnegative (i.e., all the elements are nonnegative) and have nonnegative determinants.*

(ii) *The minimum matrix multiplication ordering problem with $m \times m$ matrices with $m \ge 3$ is strongly NP-hard, even if all matrices are nonnegative and upper triangular.*

We also deal with the target version of the matrix multiplication ordering problem, i.e., minimizing the objective function $|\boldsymbol{w}^\top M_{\sigma(n)} \cdots M_{\sigma(1)} \boldsymbol{y} - t|$ for a given target $t \in \mathbb{R}$.

▶ **Theorem 7.** *Given matrices $M_1, \dots, M_n$, two vectors $\boldsymbol{w}, \boldsymbol{y}$ and a target $t \in \mathbb{R}$, the problem to decide whether there exists a permutation σ such that $|\boldsymbol{w}^\top M_{\sigma(n)} \cdots M_{\sigma(1)} \boldsymbol{y} - t| \le c_1 \cdot \min_\rho |\boldsymbol{w}^\top M_{\rho(n)} \cdots M_{\rho(1)} \boldsymbol{y} - t| + c_2$ for any positive c_1 and c_2 is strongly NP-complete.*

This means that the target version is non-approximable. We can show that the target version is also non-approximable, even if the matrices correspond to increasing linear functions.

We then consider the relationship to matrices in max-plus algebra. Let $\mathbb{R}_{\max}$ be the set $\mathbb{R} \cup \{-\infty\}$ with two binary operations max and + denoted by $\oplus$ and $\otimes$ respectively, i.e., for $a, b \in \mathbb{R}_{\max}$, $a \oplus b = \max\{a, b\}$ and $a \otimes b = a + b$. The triple $(\mathbb{R}_{\max}, \oplus, \otimes)$ is called *max-plus* *algebra*. We denote by $\mathbb{0}$ the additive identity $-\infty$, and denote by $\mathbb{1}$ the multiplicative identity 0. This notation makes it easier for us to see the correspondence between max-plus algebra and the conventional algebra. The two operations $\oplus$ and $\otimes$ are naturally extended to the matrices on $\mathbb{R}_{\max}$.

Bouquard, Lenté and Billaut [3] dealt with the problem to minimize the objective function

$$\begin{pmatrix} \mathbb{1} & \mathbb{0} & \dots & \mathbb{0} \end{pmatrix} \otimes N_{\sigma(n)} \otimes \cdots \otimes N_{\sigma(1)} \otimes \begin{pmatrix} \mathbb{0} \\ \vdots \\ \mathbb{0} \\ \mathbb{1} \end{pmatrix}, \tag{1}$$

where each N_i is an upper triangular matrix in $\mathbb{R}_{\max}^{m\times m}$. They showed that the problem in the case $m = 2$ is a generalization of the two-machine flow shop scheduling problem to minimize the makespan, and is solvable in $O(n \log n)$ time by using an extension of Johnson's rule [16] for the two-machine flow shop scheduling. Kubo and Nishinari referred to the relationship between the flow shop scheduling and the conventional matrix multiplication [19]. Focusing on this relationship, we show that the following result equivalent to the one of Bouquard et al. is obtained as a corollary of Theorem 5 (i). For a max-plus matrix $N = \begin{pmatrix} a & b \\ \mathbb{0} & d \end{pmatrix}$, where $a, b, d \neq \mathbb{0}$, we introduce $\kappa(N)$ as follows:

$$\kappa(N) = \begin{cases} (-1,\ b-a) & (a > d), \\ (\ \ 0,\ 0) & (a = d), \\ (\ \ 1,\ d-b) & (a < d). \end{cases}$$

▶ **Theorem 8.** *For the minimum max-plus matrix multiplication ordering problem with* $\boldsymbol{w} = \begin{pmatrix} \mathbb{1} & \mathbb{0} \end{pmatrix}^\top$, $\boldsymbol{y} = \begin{pmatrix} \mathbb{0} & \mathbb{1} \end{pmatrix}^\top$, *and* 2×2 *upper triangular matrices, that is, the objective function (1) in the case* $m = 2$, *a minimum permutation can be obtained in the lexicographic order for* κ.

The organization of the paper

The rest of this paper is organized as follows. Section 2 provides some notation and basic properties needed in the paper. In Section 3, we consider composition orderings for increasing linear functions and provide an outline of the proof of Theorem 1. In Section 4 we deal with general linear functions and make the exposition of ideas for an FPT algorithm to prove Theorem 4. In Section 5 we generalize composition of linear functions to matrix multiplication in the conventional algebra and max-plus algebras, and outline the proofs of Theorems 5 and 8.

2 Notation and Basic Properties

In this section, we first fix notation and present several basic properties of linear functions, which will be used in this paper. We then mention that minimum and maximum compositions are polynomially equivalent.

We view a linear function $f(x) = ax + b$ as the vector $\overrightarrow{f} = \begin{pmatrix} b \\ 1-a \end{pmatrix}$ in $\mathbb{R}^2$, and its *angle*, denoted by $\theta(f)$, is defined as the polar angle in $[0, 2\pi)$ of the vector, where we define $\theta(f) = \perp$ if the vector of f is the origin $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, i.e., f is the identity function.

For two reals ℓ and r with $\ell < r$, let $[\ell, r] = \{x \in \mathbb{R} \mid l \leq x \leq r\}$. Similarly, we denote semi-open intervals by $(\ell, r]$ and $[\ell, r)$, and open intervals by (ℓ, r). For a linear function $f(x) = ax + b$, we respectively denote by $\alpha(f)$ and $\beta(f)$ the slope and intercept of $f(x)$, i.e., $\alpha(f) = a$ and $\beta(f) = b$. A linear function f is respectively called *increasing*, *constant*, and *decreasing* if $\alpha(f) > 0$, $\alpha(f) = 0$, and $\alpha(f) < 0$. Since the result of arithmetic operations on angles may take a value outside of $[0, 2\pi)$, we provide some notation to deal with such situations, some of which have already been used in the introduction. For two angles $\theta_1, \theta_2 \in \mathbb{R}$, we write $\theta_1 =_{2\pi} \theta_2$ if they are congruent on the angle, i.e., $\theta_1 - \theta_2 \in 2\pi\mathbb{Z}$, and define $[\theta_1, \theta_2]_{2\pi} = \{\theta \in [\lambda_1, \lambda_2] \mid \lambda_1 =_{2\pi} \theta_1,\ \lambda_2 =_{2\pi} \theta_2,\ \lambda_2 - \lambda_1 \in [0, 2\pi)\}$. For example, if $\theta_1 = 3\pi/2$ and $\theta_2 = \pi/3$ then $[3\pi/2, \pi/3]_{2\pi} = \cdots \cup [-\pi/2, \pi/3] \cup [3\pi/2, 7\pi/3] \cup [7\pi/2, 13\pi/3] \cup \cdots$. We similarly define open and semi-open intervals such as $(\theta_1, \theta_2)_{2\pi}$, $[\theta_1, \theta_2)_{2\pi}$, and $(\theta_1, \theta_2]_{2\pi}$. For a non-interval set S, we define $S_{2\pi} = \{\theta \mid \theta =_{2\pi} \lambda \text{ for } \lambda \in S\}$.

We next state four basic properties of linear functions. Note that Lemmas 9, 10, and 11 do not assume increasing linear functions.

▶ **Lemma 9.** *Let g be the identity function, i.e., $g(x) = x$. Then for any function h, we have $h \circ g = g \circ h = h$.*

▶ **Lemma 10.** *For two non-identical linear functions g and h, we have the following two equivalences. The inequality for functions means that the inequality holds for any argument.*

(i) $h \circ g < g \circ h \;\Leftrightarrow\; \theta(h) - \theta(g) \in (0, \pi)_{2\pi}$.

(ii) $h \circ g = g \circ h \;\Leftrightarrow\; \theta(h) - \theta(g) \in \{0, \pi\}_{2\pi}$.

▶ **Lemma 11.** *Let g and h be two linear functions. Then $\overrightarrow{h \circ g} = \vec{h} + \alpha(h)\vec{g}$.*

▶ **Lemma 12.** *For non-identical increasing linear functions g and h, we have the following statements.*

(i) $\theta(h) - \theta(g) \in (0, \pi)_{2\pi} \Leftrightarrow \theta(h \circ g) \in (\theta(g), \theta(h))_{2\pi} \Leftrightarrow \theta(g \circ h) \in (\theta(g), \theta(h))_{2\pi}$.

(ii) $\theta(h) - \theta(g) \in \{0, \pi\}_{2\pi} \Leftrightarrow \theta(h \circ g) \in \{\theta(g), \theta(h), \perp\} \Leftrightarrow \theta(g \circ h) \in \{\theta(g), \theta(h), \perp\}$.

(iii) $\theta(h) = \theta(g) \Rightarrow \theta(h \circ g) = \theta(g \circ h) = \theta(h)\ (= \theta(g))$.

(iv) $\theta(h \circ g) = \perp \Leftrightarrow \theta(g \circ h) = \perp \Rightarrow \theta(h) - \theta(g) =_{2\pi} \pi$.

For linear functions $f_1, \ldots, f_n$ and a permutation $\sigma : [n] \to [n]$, we denote $f_{\sigma(n)} \circ \cdots \circ f_{\sigma(1)}$ by f^σ. Before ending this section, we provide a linear-time transformation between the maximization problem and the minimization problem [17]. For a linear function $f(x) = ax + b$, we define a linear function $\tilde{f}$ by

$$\tilde{f}(x) \;=\; ax - b. \tag{2}$$

Note that the slope of $\tilde{f}$ is the same as that of f. For linear functions $f_1, \ldots, f_n$ and a permutation $\sigma : [n] \to [n]$, we have $\beta(f^\sigma) = -\beta(\widetilde{f^\sigma})$. Since any permutation $\sigma : [n] \to [n]$ provides $\alpha(f^\sigma) = \alpha(\widetilde{f^\sigma}) = \prod_{i \in [n]} \alpha(f_i)$, we can see that the maximization problem with $f_1, \ldots, f_n$ is equivalent to the minimization problem with $\tilde{f}_1, \ldots, \tilde{f}_n$. Therefore, we mainly deal with the minimization problem with linear functions.

3 Composition of Increasing Linear Functions

In this section, we consider composition orderings for increasing linear functions. Especially, we provide an outline of the proof of Theorem 1.

We first prove Theorem 1 (i), which can be easily obtained from basic properties in Section 2.

Proof of Theorem 1 (i). Let us first show the only-if part. For any $i \in [n-1]$, let $\rho_i : [n] \to [n]$ be the i-th adjacent transposition, i.e., the transposition of two consecutive integers i and $i+1$. Let $\mathrm{id} : [n] \to [n]$ denote the identity permutation. Then we have $f^{\rho_i} = f^{\mathrm{id}}$, since $f_i \circ f_{i+1} = f_{i+1} \circ f_i$ by Lemmas 9 and 10 (ii). It is well-known that any permutation can be obtained by a product of adjacent transpositions and therefore for any permutation σ we obtain $f^\sigma = f^{\mathrm{id}}$, which is minimum.

For the if part, suppose, without loss of generality, that f_1 and f_2 are not collinear. Then we have $f_1 \circ f_2 \neq f_2 \circ f_1$ by Lemma 10 (ii), which implies that $f_1 \circ f_2 \circ (f_n \circ \cdots \circ f_3) \neq f_2 \circ f_1 \circ (f_n \circ \cdots \circ f_3)$, which completes the proof of the if part. ◀

Note that in fact Theorem 1 (i) does not require increasing linear functions, and hence it is true even if f_i's are general linear functions.

The next lemma plays an important role throughout the paper.

▶ **Lemma 13.** *A locally minimum permutation for non-collinear increasing linear functions is counterclockwise.*

By this and the following lemma, we can obtain the proof of Theorem 1 (ii).

▶ **Lemma 14.** *Let $\sigma : [n] \to [n]$ be a counterclockwise permutation for increasing linear functions $f_1, \ldots, f_n$. If it provides the identity, i.e., $f^\sigma(x) = x$, then any of the counterclockwise permutations provides the identity.*

Proof. Let a permutation $\tau : [n] \to [n]$ provide the identity, i.e., $f^\tau(x) = x$. By Lemma 12 (iv), $(f_{\tau(n)} \circ \cdots \circ f_{\tau(k+1)}) \circ (f_{\tau(k)} \circ \cdots \circ f_{\tau(1)}) = (f_{\tau(k)} \circ \cdots \circ f_{\tau(1)}) \circ (f_{\tau(n)} \circ \cdots \circ f_{\tau(k+1)})$ for any $k \in \{0, 1, \ldots, n-1\}$. The permutation producing the right-hand side is $\tau_{1,k,n}$, which we will denote by τ_k and call *k-shift* of τ. The equality means that the composite by τ coincides with the one by its k-shift, that is, $f^\tau = f^{\tau_k}$.

Moreover, for any permutation $\nu : [n] \to [n]$, $\theta(f_{\nu(k)}) = \theta(f_{\nu(k+1)})$ for $k \in [n-1]$ implies that $f^\nu = f^{\nu_{k,k,k+1}}$ by Lemma 10 (ii).

Since any of the counterclockwise permutations is obtained by repeatedly applying adjacent transpositions for the same angles and k-shift of σ, the two claims provide the proof. ◀

Proof of Theorem 1 (ii). (ii-1) $\Longrightarrow$ (ii-2) follows from Lemmas 13 and 14.

For the converse direction, by Lemma 13 we suppose, on the contrary, that all counterclockwise permutations provide the same non-identical function g. Since f_i's are not collinear, there exists a non-identical linear function f_i such that

$$\theta(f_i) \quad \notin \{\theta(g), \theta(g) + \pi\}_{2\pi}. \tag{3}$$

Consider a counterclockwise permutation $\sigma : [n] \to [n]$ with $\sigma(1) = i$, and let $h = f_{\sigma(n)} \circ \cdots \circ f_{\sigma(2)}$. Then we have $g = h \circ f_i$. Since $\theta(h) \notin \{\theta(f_i), \theta(f_i) + \pi\}_{2\pi} \cup \{\perp\}$ by (3) and Lemma 11, Lemma 10 (i) implies that $h \circ f_i \neq f_i \circ h$, which contradicts the assumption. ◀

Example 15 demonstrates the optimal condition in Theorem 1 (iii). Since the proof of Theorem 1 (iii) is also involved, we only mention that it relies on the *unimodality* of f^σ for counterclockwise permutations σ.

▶ **Example 15.** Consider the following five increasing linear functions:

$$f_1 = \frac{1}{2}x + 1, \;\; f_2 = \frac{1}{3}x - 1, \;\; f_3 = 2x - 2, \;\; f_4 = 2x - 1, \;\text{ and } \; f_5 = 3x.$$

Then their vectors are given as follows (See Figure 2):

$$\vec{f_1} = \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}, \ \vec{f_2} = \begin{pmatrix} -1 \\ 2/3 \end{pmatrix}, \ \vec{f_3} = \begin{pmatrix} -2 \\ -1 \end{pmatrix}, \ \vec{f_4} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \text{ and } \vec{f_5} = \begin{pmatrix} 0 \\ -2 \end{pmatrix}.$$

Note that the identity permutation id : $[n] \to [n]$ is counterclockwise for f_i's, and moreover, by Lemma 13, we can see that it is minimum, since

$$f^{\mathrm{id}} = 2x - 23,\ f^{\mathrm{id}_1} = 2x - \frac{27}{2},\ f^{\mathrm{id}_2} = 2x - \frac{19}{6},\ f^{\mathrm{id}_3} = 2x - \frac{13}{3},\ f^{\mathrm{id}_4} = 2x - \frac{23}{3},$$

which also shows that $(f^{\mathrm{id}}, f^{\mathrm{id}_1}, f^{\mathrm{id}_2}, f^{\mathrm{id}_3}, f^{\mathrm{id}_4})$ is unimodal.

We can also see that the identity permutation satisfies $\theta(f^{\mathrm{id}}) + \pi \in [\theta(f_5), \theta(f_1)]_{2\pi}$.

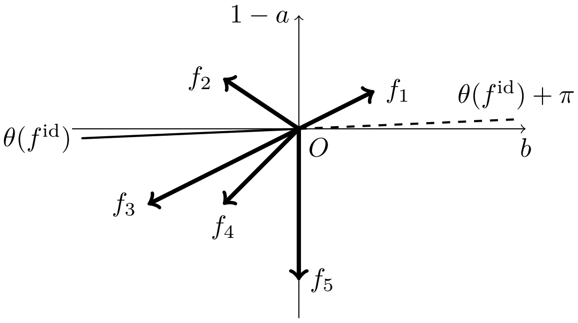


Figure 2 The vector representation for $f_1, \ldots, f_5$.

▶ Remark 16. As discussed in Section 2, the maximization for f_i's is equivalent to the minimization for $\widetilde{f_i}$'s given by (2). Thus all the results for increasing functions are applicable for the maximization problem. Since the transformation (2) is the reflection across the $(1-a)$-axis in the vector representation, we can obtain the results by exchanging the term "counterclockwise" by "clockwise".

Corollary 2 is an immediate and direct conclusion of Theorem 1. Theorem 3 is proved by using Theorem 1.

We can generalize increasing linear functions to nondecreasing linear functions in Theorems 1 and 3, and Corollary 2.

4 Composition of General Linear Functions

In this section, we discuss the composition of general linear functions $f_1, \ldots, f_n$, where an example of composition for general linear functions is given in Example 17. Let k denote the number of decreasing functions in them, i.e., $k = |\{i \in [n] \mid \alpha(f_i) < 0\}|$. In Section 3 we provided structural characterizations for the minimum permutations when $k = 0$. We present several structural properties for minimum permutations for general linear functions and show ideas for FPT with respect to k for the minimization problem, whose complexity status was open [17].

In the rest of this section, we restrict our attention to the case where no linear function is identity or constant, i.e., $f_i(x) \neq x$ and $\alpha(f_i) \neq 0$ for all $i \in [n]$. Note that the identity function plays no role in minimum composition. For a constant function $f(x) = b$, we consider $f^{(\epsilon)}(x) = \epsilon x + b$ for some $\epsilon > 0$ (we set $f^{(\epsilon)} = f$ for a non-constant function) and can reduce the case containing constant functions to the case of increasing functions. In other words, we can show that the minimality for $f_1^{(\epsilon)}, \dots, f_n^{(\epsilon)}$ implies the one for $f_1, \dots, f_n$, if $|\epsilon|$ is sufficiently small. We remark that our algorithm does not make use of ϵ explicitly, since the orderings of angles $\theta(f_i^{(\epsilon)})$'s are only needed.

▶ **Example 17.** Consider the following seven linear functions:

$$f_1 = \frac{1}{3}x,\ f_2 = \frac{2}{3}x + 1,\ f_3 = x + \frac{1}{2},\ f_4 = -x - 3,\ f_5 = x - 1,\ f_6 = \frac{3}{2}x,\ f_7 = 2x + 1.$$

All but f_4 are increasing. The vector representation is shown in Figure 3.

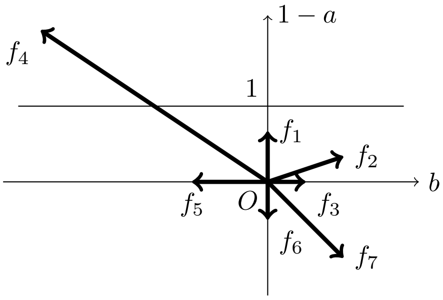


Figure 3 The vector representation for $f_1, \dots, f_7$.

Note that the identity permutation is minimum. Recall that $\theta(f_{\sigma(i+1)}) - \theta(f_{\sigma(i)}) \in [0, \pi]_{2\pi}$ holds for any minimum permutation σ for increasing linear functions by Theorem 3 and Lemma 10. However, this crucial property for increasing linear functions does not hold in general. For example, $\theta(f_2) - \theta(f_1) \in (\pi, 2\pi)_{2\pi}$. Instead, we point out the following properties: $f_3 \circ f_2 \circ f_1$ before f_4 is provided by a maximum permutation for f_1, f_2, and f_3, while $f_7 \circ f_6 \circ f_5$ after f_4 is provided by a minimum permutation for f_5, f_6, and f_7. We also note that f_4 is not suitable for processing time, since both coefficients are negative.

We define two sets L^σ and U^σ of increasing linear functions. For a permutation $\sigma : [n] \to [n]$, let $n_1^\sigma, \dots, n_k^\sigma$ be integers such that $n_1^\sigma < \cdots < n_k^\sigma$ and $\alpha(f_{\sigma(n_j^\sigma)}) < 0$ for all $j \in [k]$. For $j \in \{0, 1, \dots, k\}$, let $I_j^\sigma = \{i \in [n] \mid n_j^\sigma < i < n_{j+1}^\sigma\}$, where $n_0^\sigma = 0$ and $n_{k+1}^\sigma = n + 1$, and define

$$L^\sigma = \bigcup_{k-j:\text{even}} I_j^\sigma \quad \text{and} \quad U^\sigma = \bigcup_{k-j:\text{odd}} I_j^\sigma.$$

By definition, the set of indices of all increasing functions $\{i \in [n] \mid \alpha(f_{\sigma(i)}) \geq 0\}$ is partitioned into L^σ and U^σ. In Example 17, we have $L^{\text{id}} = I_1^{\text{id}} = \{5, 6, 7\}$ and $U^{\text{id}} = I_0^{\text{id}} = \{1, 2, 3\}$.

The following lemma states that L^σ and U^σ are permuted counterclockwisely and clockwisely, respectively, if σ is minimum. Let $L^\sigma = \{\ell_1, \dots, \ell_{|L^\sigma|}\}$ and $U^\sigma = \{u_1, \dots, u_{|U^\sigma|}\}$, where $\ell_1 < \cdots < \ell_{|L^\sigma|}$ and $u_1 < \cdots < u_{|U^\sigma|}$, and let

$$p_i = f_{\sigma(\ell_i)} \quad \text{for} \quad i \in [|L^\sigma|] \quad \text{and} \quad q_i = f_{\sigma(u_i)} \quad \text{for} \quad i \in [|U^\sigma|].$$

▶ **Lemma 18.** *Let $\sigma : [n] \to [n]$ be a minimum permutation for non-constant and non-identical linear functions $f_1, \dots, f_n$. Let p_i $(i \in [|L^\sigma|])$ and q_i $(i \in [|U^\sigma|])$ denote increasing linear functions defined as above. Then we have the following two statements.*

(i) *The identity* id : $[|L^\sigma|] \to [|L^\sigma|]$ *is counterclockwise for p_i's, unless they are collinear.*

(ii) *The identity* id : $[|U^\sigma|] \to [|U^\sigma|]$ *is clockwise for q_i's, unless they are collinear.*

Outline of Proof. We only prove the case where k is even and (i), since the odd case or (ii) can be treated similarly.

$$f^\sigma = \overbrace{p_{|L^\sigma|} \circ \cdots \circ p_{|L^\sigma|-|I^\sigma_k|+1}}^{I^\sigma_k} \circ g_{k/2} \circ \cdots \circ g_2 \circ \overbrace{p_{|I^\sigma_0|+|I^\sigma_2|} \circ \cdots \circ p_{|I^\sigma_0|+1}}^{I^\sigma_2} \circ g_1 \circ \overbrace{p_{|I^\sigma_0|} \circ \cdots \circ p_1}^{I^\sigma_0}$$

where $g_j = f_{\sigma(n^\sigma_{2j})} \circ f_{\sigma(n^\sigma_{2j}-1)} \circ \cdots \circ f_{\sigma(n^\sigma_{2j-1})}$ for $j \in \{1, \ldots, \lceil k/2 \rceil\}$,and we set $f_{\sigma(n+1)} = f_{\sigma(0)} = x$. Note that only two linear functions at both ends are decreasing.

Since all the linear functions in the right-hand side are increasing, Theorem 1 implies (i) of the lemma. ◀

Moreover, the following crucial lemma (iii) shows that L^σ and U^σ are partitioned by two angles ψ_1 and ψ_2. For an set $I \subseteq [n]$, let $\theta(I) = \{\theta(f_{\sigma(i)}) \mid i \in I\}$.

▶ **Lemma 19.** *There exists a minimum permutation $\sigma : [n] \to [n]$ for non-constant and non-identical linear functions $f_1, \ldots, f_n$ such that*

- **(i)** *$f_{\sigma(\ell)}$ ($\ell \in L^\sigma$) are permuted counterclockwisely,*
- **(ii)** *$f_{\sigma(u)}$ ($u \in U^\sigma$) are permuted clockwisely,*
- **(iii)** *$\theta(L^\sigma) \subseteq [\psi_1, \psi_2]$ and $\theta(U^\sigma) \subseteq (\psi_2, \psi_1)_{2\pi}$ for some two angles $\psi_1 \in (0, \pi)$ and $\psi_2 \in (\pi, 2\pi)$,*
- **(iv)** *$\theta(I^\sigma_s) \cap \theta(I^\sigma_t) = \emptyset$ for any distinct s and t.*

This directly implies that a minimum permutation for linear functions $f_1, \ldots, f_n$ can be computed in $O(k!\, n^{k+4})$ time, where k denotes the number of decreasing f_i's. The reason is as follows. Assume first that no f_i is identity and we utilize $f^{(\epsilon)}_i$'s instead of f_i's. By Lemma 19 (iii), we essentially have n^2 possible angles ψ_1 and ψ_2. Based on such angles, we partition the set of indices of increasing linear functions into $I_0, \ldots, I_k$. By Lemma 19 (i), (ii), and (iv), we have at most n^{k+1} many such partitions. Since there exist $k!$ orderings of decreasing functions, by checking at most $k!\, n^{k+3} (= n^2 \times n^{k+1} \times k!)$ permutations σ, we obtain a minimum permutation for f_i's. Note that each such permutation σ and the composite f^σ can be computed in $O(n)$ time, after sorting $\theta(f^{(\epsilon)}_i)$'s. Since $\theta(f^{(\epsilon)}_s)$ and $\theta(f^{(\epsilon)}_t)$ can be compared in $O(1)$ time for sufficiently small $\epsilon > 0$ without exactly computing their angles, we can sort $\theta(f^{(\epsilon)}_i)$'s in $O(n \log n)$ time. Thus in total we require $O(k!\, n^{k+4} + n \log n) = O(k!\, n^{k+4})$ time. If some f_i's are identities, then we can put them into I_0, where I_0 is obtained in the procedure above for the non-identical functions. Therefore, a minimum permutation can be computed in $O(k!\, n^{k+4})$ time.

In order to improve this XP result, namely, to have an FPT algorithm with respect to k, we apply the dynamic programming approach to the following problem.

Problem LU-Ordered Minimum Composition

Input: Two sets of increasing linear functions $L = \{p_1, \ldots, p_{|L|}\}$ and $U = \{q_1, \ldots, q_{|U|}\}$, and decreasing linear functions $g_1, \ldots, g_k$ with $k > 0$.

Output: A minimum permutation σ for linear functions in $L \cup U \cup \{g_1, \ldots, g_k\}$ such that

- **(i)** $L^\sigma = L$ and $U^\sigma = U$,
- **(ii)** the restriction of σ on L produces the ordering $(p_1, \ldots, p_{|L|})$, and
- **(iii)** the restriction of σ on U produces the ordering $(q_1, \ldots, q_{|U|})$.

Note that a minimum permutation for the original problem can be computed by solving **Problem** LU-Ordered Minimu Composition $O(n^4)$ times for $|L| + |U| \leq n - k$. Since the problem can be solved in $O(2^k k(|L| + |U| + k)^2)$ time, we obtain Theorem 4.

5 Matrix Multiplication

In this section, we consider matrix multiplication orderings as a generalization of composition orderings for linear functions. We provide outlines of the proofs of Theorems 5 and 8, and refer to the problems we use to prove Theorems 6 and 7.

In order to prove Theorem 5, we first assume that matrices $M_1, \dots, M_n$ in $\mathbb{R}^{2\times 2}$ are all upper triangular. Then we have the following lemma.

▶ **Lemma 20.** *The minimum matrix multiplication ordering problem with 2×2 upper triangular matrices can be reduced to the one with $\boldsymbol{w}^\top = \begin{pmatrix} 1 & 0 \end{pmatrix}$, $\boldsymbol{y}^\top = \begin{pmatrix} 0 & 1 \end{pmatrix}$, and 2×2 upper triangular matrices with positive $(2,2)$-entries.*

Thus we can assume that a given upper triangular matrix $M_i = \begin{pmatrix} a_i & b_i \\ 0 & d_i \end{pmatrix}$ has a positive d_i for $i \in [n]$. We then have

$$\begin{pmatrix} 1 & 0 \end{pmatrix} M^\sigma \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \left(\prod_{i=1}^{n} d_i \right) f^\sigma(0),$$

where $f_i(x) = (a_i/d_i)x + b_i/d_i$ for $i \in [n]$. This implies that the minimum matrix multiplication ordering problem with 2×2 upper triangular matrices can be solved by solving the minimum composition ordering problem with linear functions. We remark that our algorithm concerns the comparison of polar angles $\theta(f_i)$'s, but not of the vectors $\begin{pmatrix} b_i/d_i \\ 1 - a_i/d_i \end{pmatrix}$, and hence we do not need to care about the case where $d_i = \epsilon$. Therefore, we have the following lemma.

▶ **Lemma 21.** *For the minimum matrix multiplication ordering problem with n 2×2 upper triangular matrices, we have the following statements.*

(i) *If all matrices have nonnegative determinants, then a minimum permutation can be computed in $O(n \log n)$ time.*

(ii) *If some matrix has a negative determinant, then a minimum permutation can be computed in $O(k2^k n^6)$ time, where k denotes the number of matrices with negative determinants.*

This immediately implies Theorem 5.

Unfortunately, this positive results cannot be extended to 1) the nonnegative determinant case for $m = 2$, 2) the case of $m \geq 3$, and 3) the target version; see Theorem 6 (i), (ii) and Theorem 7. We use the 3-partition problem to prove Theorems 6 (i) and 7.

Bouquard et al. [3] showed that the problem to minimize (1) for the case $m \geq 3$ is strongly NP-hard by reduction from the three-machine flow shop scheduling problem to minimize the makespan, which is known to be strongly NP-hard [7]. We use the former problem to prove Theorem 6 (ii).

References

1 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In Moses Charikar, Klaus Jansen, Omer Reingold, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 16–28, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74208-1_2.

2 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA07, pages 434–443, USA, 2007. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283429`.

3 J.-L. Bouquard, C. Lenté, and J.-C. Billaut. Application of an optimization problem in max-plus algebra to scheduling problems. *Discrete Applied Mathematics*, 154(15):2064–2079, 2006. `doi:10.1016/j.dam.2005.04.011`.

4 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA05, pages 395–404, USA, 2005. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=1070432.1070487`.

5 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008. `doi:10.1287/MOOR.1080.0330`.

6 Thomas S. Ferguson. Who Solved the Secretary Problem? *Statistical Science*, 4(3):282–289, 1989. `doi:10.1214/ss/1177012493`.

7 Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976. `doi:10.1287/moor.1.2.117`.

8 Stanisław Gawiejnowicz. *Models and Algorithms of Time-Dependent Scheduling*. Springer, 2 edition, 2020. `doi:10.1007/978-3-662-59362-2`.

9 Stanisław Gawiejnowicz. A review of four decades of time-dependent scheduling: main results, new topics, and open problems. *Journal of Scheduling*, 23:3–47, February 2020. `doi:10.1007/s10951-019-00630-w`.

10 Stanisław Gawiejnowicz and Wiesław Kurc. New results for an open time-dependent scheduling problem. *Journal of Scheduling*, 23:733–744, December 2020. `doi:10.1007/s10951-020-00662-7`.

11 Stanisław Gawiejnowicz, Wiesław Kurc, and Lidia Pankowska. Conjugate problems in time-dependent scheduling. *Journal of Scheduling*, 12:543–553, October 2009. `doi:10.1007/s10951-009-0121-0`.

12 Stanisław Gawiejnowicz and Bertrand M.T. Lin. Scheduling time-dependent jobs under mixed deterioration. *Applied Mathematics and Computation*, 216:438–447, March 2010. `doi:10.1016/j.amc.2010.01.037`.

13 Stanisław Gawiejnowicz and Lidia Pankowska. Scheduling jobs with varying processing times. *Information Processing Letters*, 54(3):175–178, 1995. `doi:10.1016/0020-0190(95)00009-2`.

14 Jatinder N.D. Gupta and Sushil K. Gupta. Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387–393, 1988. `doi:10.1016/0360-8352(88)90041-1`.

15 Kevin I-J. Ho, Joseph Y-T. Leung, and W-D. Wei. Complexity of scheduling tasks with time-dependent execution times. *Information Processing Letters*, 48(6):315–320, 1993. `doi:10.1016/0020-0190(93)90175-9`.

16 Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954. `doi:10.1002/nav.3800010110`.

17 Yasushi Kawase, Kazuhisa Makino, and Kento Seimi. Optimal composition ordering problems for piecewise linear functions. *Algorithmica*, 2018. `doi:10.1007/s00453-017-0397-y`.

18 Susumu Kubo, Kazuhisa Makino, and Souta Sakamoto. Composition orderings for linear functions and matrix multiplication orderings. *arXiv*, 2024. `arXiv:2402.10451`, `doi:10.48550/arXiv.2402.10451`.

19 Susumu Kubo and Katsuhiro Nishinari. Applications of max-plus algebra to flow shop scheduling problems. *Discrete Applied Mathematics*, 247:278–293, 2018. `doi:10.1016/j.dam.2018.03.045`.

20 Gur Mosheiov. Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, 21(6):653–659, 1994. doi:10.1016/0305-0548(94)90080-9.

21 Krzysztof M. Ocetkiewicz. Partial dominated schedules and minimizing the total completion time of deteriorating jobs. *Optimization*, 62:1341–1356, October 2013. doi:10.1080/02331934.2013.836647.

22 Shayan Oveis Gharan and Jan Vondrák. On variants of the matroid secretary problem. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, pages 335–346, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-23719-5_29.

23 V. S. Tanaev, V. S. Gordon, and Y. M. Shafransky. *Scheduling Theory. Single-Stage Systems*. Springer, 1994.

24 Wiesław Wajs. Polynomial algorithm for dynamic sequencing problem. *Archiwum Automatyki i Telemechaniki*, 31:209–213, January 1986.

A Simple Distributed Algorithm for Sparse Fractional Covering and Packing Problems

Qian Li
Shenzhen International Center For Industrial And Applied Mathematics,
Shenzhen Research Institute of Big Data, China

Minghui Ouyang
School of Mathematical Sciences, Peking University, Beijing, China

Yuyi Wang
Lambda Lab, China Railway Rolling Stock Corporation Zhuzhou Institute, China

Abstract

This paper presents a distributed algorithm in the CONGEST model that achieves a $(1+\epsilon)$-approximation for row-sparse fractional covering problems (RS-FCP) and the dual column-sparse fraction packing problems (CS-FPP). Compared with the best-known $(1+\epsilon)$-approximation CONGEST algorithm for RS-FCP/CS-FPP developed by Kuhn, Moscibroda, and Wattenhofer (SODA'06), our algorithm is not only much simpler but also significantly improves the dependency on ϵ.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases CONGEST model, row-sparse fractional covering, column-sparse fractional packing, positive linear programming, simple algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.45

Related Version *Full Version*: https://arxiv.org/abs/2409.16168

Funding Qian Li's work is supported by the National Key R&D Program of China under Grant 2022YFA1003900, the Hetao Shenzhen-Hong Kong Science and Technology Innovation Cooperation Zone Project (No.HZQSWS-KCCYB-2024016), and the National Natural Science Foundation of China Grants No.62002229. Minghui Ouyang's work is supported in part by the National Natural Science Foundation of China Grants No.62002229. Yuyi Wang's work is supported by the Natural Science Foundation of Hunan Province 2024JJ5128.

1 Introduction

A *fractional covering* problem (FCP) and its dual *fractional packing* problem (FPP) are positive linear programs (LP) of the canonical form:

$$\begin{array}{llll} \min\limits_{\boldsymbol{x}} & \boldsymbol{c}^T\boldsymbol{x} \quad \text{(FCP, the primal LP)} & \max\limits_{\boldsymbol{y}} & \boldsymbol{b}^T\boldsymbol{y} \quad \text{(FPP, the dual LP)} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} & \text{s.t.} & \boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{c} \\ & \boldsymbol{x} \geq 0 & & \boldsymbol{y} \geq 0, \end{array}$$

where all A_{ij}, b_i, and c_j are non-negative. This paper particularly focuses on k-row-sparse FCPs (k-RS-FCP) and k-column-sparse FPPs (k-CS-FPP). These are FCPs and FPPs in which the matrix $\boldsymbol{A}$ contains at most k non-zero entries per row. They are still fairly general problems and can model a broad class of basic problems in combinatorial optimization, such as the fractional version of vertex cover, bounded-frequency weighted set cover, weighted k-uniform hypergraph matching, stochastic matching, and stochastic k-set packing.

This paper studies distributed algorithms for FCPs and FPPs in the CONGEST model. The CONGEST model features a network $G = (V, E)$, where each node corresponds to a processor and each edge (u, v) represents a bidirectional communication channel between

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 45; pp. 45:1–45:8
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

processors u and v. The computation proceeds in rounds. In one round, each processor first executes local computations and then sends messages to its neighbors. Each of the messages is restricted to $O(\log |V|)$ bits. The algorithm complexity is measured in the number of rounds it performs.

For an FCP/FPP instance with m primal variables and n dual variables (i.e., A has dimensions $n \times m$), the network is a bipartite graph $G = ([m], [n], E)$. Each primal variable x_j is associated with a left node $j \in [m]$, and each dual variable y_i is associated with a right node $i \in [n]$. An edge (i, j) exists if and only if $A_{ij} > 0$. At the beginning of a CONGEST algorithm, each left node j only knows its corresponding cost c_j and the column vector $(A_{ij} : i \in [n])$, and each right node i only knows b_i and the row vector $(A_{ij} : j \in [m])$. At the end of the algorithm, each left node j is required to output a number $\hat{x}_j$ and each right node i to output a $\hat{y}_i$, which together are supposed to form approximate solutions to the FCP and FPP respectively.

Let $\mathbf{1}_k$ denote the k-dimensional all-ones vector. The subscript k will be dropped if it is implicit. Without loss of generality, this paper considers FCP and FPP instances of the following normal forms:

$$\min_{\boldsymbol{x}} \quad \mathbf{1}^T\boldsymbol{x} \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \geq \mathbf{1} \text{ and } \boldsymbol{x} \geq 0 \tag{1}$$

and

$$\max_{\boldsymbol{y}} \quad \mathbf{1}^T\boldsymbol{y} \quad \text{s.t.} \quad \boldsymbol{A}^T\boldsymbol{y} \leq \mathbf{1} \text{ and } \boldsymbol{y} \geq 0 \tag{2}$$

where A_{ij} is either 0 or ≥ 1. The reduction to the normal form proceeds as follows:

- First, we can assume that each $b_i > 0$ since otherwise we can set y_i to zero and delete the i-th row of A;
  similarly, we can assume each $c_j > 0$ since otherwise we can set x_j to $+\infty$ and delete the j-th column of A, and then set y_i to zero and delete the i-th row for all i with $A_{ij} > 0$.
- Then, we can replace A_{ij} by $\hat{A}_{ij} = \frac{A_{ij}}{b_i c_j}$, replace $\boldsymbol{b}$ and $\boldsymbol{c}$ by all-ones vector, and work with variables $\hat{x}_j = c_j x_j$ and $\hat{y}_i = b_i y_i$.
- Finally, we replace $\hat{A}_{ij}$ with $\tilde{A}_{ij} = \frac{\hat{A}_{ij}}{\min\{\hat{A}_{i'j'} | \hat{A}_{i'j'>0}\}}$ and work with $\tilde{x}_j = \hat{x}_j \cdot \min\{\hat{A}_{i'j'} \mid \hat{A}_{i'j'>0}\}$ and $\tilde{y}_i = \hat{y}_i \cdot \min\{\hat{A}_{i'j'} \mid \hat{A}_{i'j'>0}\}$.

Papadimitriou and Yannakakis [12] initiated the research on approximating FCPs/FPPs in the CONGEST model. Bartal, Byers, and Raz [3] proposed the first constant approximation ratio algorithm with polylog$(m + n)$ rounds. After designing a distributed algorithm for a specific FCP/FPP scenario, namely the fractional dominating set problem [10], Kuhn, Moscibroda, and Wattenhofer [8] finally developed an efficient $(1+\epsilon)$-approximation algorithm for general FCP/FPP instances, running in $O(\log \Gamma_p \cdot \log \Gamma_d / \epsilon^4)$ round for normalized instances where

$$\Gamma_p \coloneqq \max_j \sum_{i=1}^{n} A_{ij}, \text{ and } \Gamma_d \coloneqq \max_i \sum_{j=1}^{m} A_{ij}.$$

Particularly, for RS-FCPs/CS-FPPs, the round complexity becomes $O\left(\log A_{\max} \cdot \log \Gamma_p / \epsilon^4\right)$. Later, Awerbuch and Khandekar [2] proposed another $(1 + \epsilon)$-approximation algorithm for general normalized FCP/FPP instances running in $\tilde{O}(\log^2(nA_{\max}) \log^2(nmA_{\max})/\epsilon^5)$ rounds, which has worse bound than [8] but enjoys the features of simplicity and statelessness.

Several works studied the lower bound for CONGEST algorithms to approximate linear programming. Bartal, Byers, and Raz [3] showed that $(1 + \epsilon)$-approximation algorithms for general FCPs/FPPs require at least $\Omega(1/\epsilon)$ rounds. Kuhn, Moscibroda, and Wattenhofer proved that [5, 6, 7, 8, 11] no constant round, constant-factor approximation CONGEST algorithms exist for the LP relaxation of minimum vertex cover, minimum dominating set, or maximum matching in general graphs. Later, they improved this lower bound by showing that [9] no $o(\sqrt{\log(m+n)/\log\log(m+n)})$ rounds CONGEST algorithms can constant-factor approximate the LP relaxation of minimum vertex cover, maximum matching, or by extension, the general binary RS-FCPs or CS-FPPs (i.e., $A_{ij} \in \{0,1\}$) as well.

In this paper, we propose a CONGEST algorithm (Algorithm 1) for approximating general RS-FCP/CS-FPP instances, which is much simpler than the algorithm of [8]. Moreover, our algorithm exhibits a worse dependency on $A_{\max}$ but improves the dependency on ϵ. In particular, for the binary RS-FCPs or CS-FPPs, which include LP relaxations of many combinatorial problems such as minimum vertex cover, minimum dominating set, maximum matching, and maximum independent set, our algorithm runs in $O(\log \Gamma_p/\epsilon^2)$ rounds, which is a $1/\epsilon^2$ factor improvement over the algorithm of [8].

▶ **Theorem 1** (Main Theorem). *For any $\epsilon > 0$, Algorithm 1 computes $(1+\epsilon)$-approximate solutions to RS-FCP and CS-FPP at the same time, running in $O(A_{\max} \cdot \log \Gamma_p/\epsilon^2)$ rounds.*

▶ Remark 2. In 2018, Ahmadi et al. [1] proposed a simple $(1+\epsilon)$-approximation distributed algorithm for the LP relaxation of minimum vertex cover and maximum weighted matching problems, which are special classes of 2-RS-FCPs and 2-CS-FPPs. They claimed the algorithm runs in $O(\log \Gamma_p/\epsilon^2)$ rounds, an $O(A_{\max})$ factor faster than our algorithm. Unfortunately, there is a flaw in their proof[1], and we do not know how to correct the proof to achieve the claimed bound.

2 Algorithms

In this section, we present our algorithm (Algorithm 1) and the analysis. Indeed, our algorithm applies to general FCP/FPP instances, and we will prove the following theorem:

▶ **Theorem 3.** *For any $\epsilon > 0$, Algorithm 1 computes $(1+\epsilon)$-approximate solutions to (1) and (2) at the same time, running in $O\left(\Gamma_d \cdot \log \Gamma_p/\epsilon^2\right)$ rounds.*

Our algorithm is based on the sequential fractional set cover algorithm by Eisenbrand et al. [4] and the fractional weighted bipartite matching by Ahmadi et al. [1]. It will be helpful to view (1) as a generalization of the fractional set cover problem. Specficially, there is a universe $U = \{e_1, \cdots, e_n\}$ of n elements, a collection $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ of subsets of U, and a matrix $\{A_{eS} : e \in U, S \in \mathcal{S}\}$ indicating the covering efficiency of S on e. We say $e \in S$ if $A_{eS} > 0$. Then (1) can be recast as the following generalization of the fractional set cover problem:

$$\min_{\boldsymbol{x} \geq 0} \quad \sum_{S \in \mathcal{S}} x_S \quad \text{s.t.} \quad \sum_{S \ni e} A_{eS} \cdot x_S \geq 1 \text{ for any } e. \tag{3}$$

The dual (2) can be recast as:

$$\max_{\boldsymbol{y} \geq 0} \quad \sum_{e \in U} y_e \quad \text{s.t.} \quad \sum_{e \in S} A_{eS} \cdot y_e \leq 1 \text{ for any } S. \tag{4}$$

[1] In the proof of Lemma 5.2 in the full version, Y_v^+ should be defined as y_e^+/w_e rather than y_e^+; $\frac{\alpha^{1/w_e}}{\alpha^{1/w_e}-1} \leq \frac{\alpha}{\alpha-1}$ seems doubtful since $\frac{1}{w_e} > 1$ in their setting.

Algorithm 1 An $(1+\epsilon)$-approximation algorithm for the normalized FCP/FPP.

```
Parameter: $\alpha, f \in \mathbb{R}_{\geq 0}$ and $L \in \mathbb{N}$ defined as in (5);
Initialize $x_S := 0$ for any $S$, and $y_e := 0$ and $r_e := 1$ for any $e$;
for $\ell = 1$ to $L$ do
    for each all $S$ in parallel do
        if $\rho_S \geq \frac{1}{\alpha} \cdot \max_{S' \cap S \neq \emptyset} \rho_{S'}$ then
            $x_S := x_S + 1$;
            for all $e \in S$ do
                $y_e := y_e + A_{eS} \cdot r_e / \left(\sum_{e \in S} A_{eS} \cdot r_e\right)$ and $r_e := r_e / \alpha^{A_{eS}}$;
                if $r_e \leq \alpha^{-f}$ then $r_e := 0$;
Return: $\boldsymbol{x}/f$ and $\boldsymbol{y}/(f \cdot (1+\epsilon))$ as the approximate solutions to (3) and (4)
    respectively.
```

Our algorithm maintains a variable x_S for each subset S, and two variables y_e and r_e for each element e. x_S and y_e are initially 0 and their values can only increase throughout the algorithm; the variable r_e is initially 1 and its value can only decreases. Intuitively, r_e denotes the "requirement" of element e. Furthermore, we define the efficiency of S as $\rho_S := \sum_{e \in S} A_{eS} \cdot r_e$.

Our algorithm consists of L phases. α and f are two other algorithmic parameters. The values of L, α, f will be determined later. In the ℓ-th phase, the algorithm picks all subsets S with

$$\rho_S \geq \frac{1}{\alpha} \cdot \max_{S' \cap S \neq \emptyset} \rho_S'.$$

and update the primal variable

$$x_S := x_S + 1,$$

as well as: for each $e \in S$,

$$y_e := y_e + \frac{A_{eS} \cdot r_e}{\sum_{e \in S} A_{eS} \cdot r_e}, \text{ and } r_e := r_e / \alpha^{A_{eS}}.$$

In other words, let $\Xi_\ell(e) := \{S \ni e \mid S \text{ is selected in the } \ell\text{-th phase}\}$, then after the ℓ-th phase, we have

$$y_e := y_e + \Delta y_e = y_e + \sum_{S \in \Xi_\ell(e)} \frac{A_{eS} \cdot r_e}{\sum_{e \in S} A_{eS} \cdot r_e}, \text{ and } r_e := r_e / \alpha^{\sum_{S \in \Xi_\ell(e)} A_{eS}}.$$

Besides, we set $r_e = 0$ as soon as $r_e \leq \alpha^{-f}$. Finally, the algorithm returns $\boldsymbol{x}/f$ and $\boldsymbol{y}/\left((1+\epsilon) \cdot f\right)$ as the approximation solutions. See Algorithm 1 for a formal description.

▶ Remark 4. The two algorithms in [8] and [1] both have a similar greedy fashion: it starts with all x_S set to 0, always increases the x_S whose "efficiency" is maximum up to a certain factor, and then distributes the increment of x_S among its elements and decreases the requirements r_e. Our algorithm and the two algorithms of [8] and [1] differ in specific implementations: the definition of efficiency, the distribution of increments, and the reduction of requirements. In particular, the algorithm of [8] consists of two levels of loops: the goal of the first-level loop is to reduce the maximum "weighted primal degree", and one complete run of the second-level loop can be seen as one parallel greedy step. This two-level structure complicates the algorithm of [8]. The algorithm of [1] only works for the LP relaxation of minimum vertex cover and maximum weighted matching problems, which are special classes of 2-RS-FCPs and 2-CS-FPPs. The distribution of increments and the reduction of requirements in our implementation is similar to [1], and the main difference is the definition of efficiency.

Before analyzing the correctness and efficiency of the algorithm, we present some helpful observations about its behavior.

▶ **Proposition 5.** *Throughout the algorithm, we always have*

(a) $\sum_S x_S = \sum_e y_e$.

(b) *For any S, the value of ρ_S is non-increasing, and lies within $\left(\alpha^{-f}, \Gamma_p\right] \cup \{0\}$.*

(c) *After each phase,* $\max_S \rho_S$ *decreases by a factor of at least α.*

Proof.

Part (a). Initially, $\sum_S x_S = \sum_e y_e = 0$. Then, whenever we increase x_S by 1, we increase $\sum_e y_e$ by $\sum_{e \in S} \frac{A_{eS} \cdot r_e}{\sum_{e \in S} A_{eS} \cdot r_e} = 1$.

Part (b). Since r_e is non-increasing, so is $\rho_S := \sum_{e \in S} A_{eS} \cdot r_e$. Besides, the initial value of ρ_S is $\sum_{e \in S} A_{eS}$, which is upper-bounded by Γ_p. Furthermore, for any non-zero r_e where $e \in S$, it should be strictly greater than α^{-f}, since otherwise it will be set to 0. Recalling that $A_{eS} \geq 1$, we have $\rho_S > \alpha^{-f}$ or $\rho_S = 0$.

Part (c). Define $\rho_{\max} := \max_S \rho_S$. Note that every S with $\rho_S \geq \rho_{\max}/\alpha$ will be picked. Then for any such S, r_e will decrease by a factor of $\alpha^{\sum_{S' \in \Xi_\ell(e)} A_{eS}} \geq \alpha^{A_{eS}} \geq \alpha$ for any $e \in S$, so $\rho_S = \sum_{e \in S} A_{eS} \cdot r_e$ decreases by a factor of at least α as well. ◀

By Proposition 5 (b) and (c), it is easy to see that after $\lceil \log_\alpha \Gamma_p + f \rceil$ phases, all ρ_S and r_e will become zero. We choose [2]

$$\alpha := 1 + \frac{\epsilon}{c \cdot \Gamma_d}, \text{ and } f := \frac{2}{\epsilon \cdot \ln \alpha} \cdot \ln \Gamma_p, \text{ and } L := \lceil \log_\alpha \Gamma_p + f \rceil. \tag{5}$$

where c is a sufficiently large constant. Note that each phase can be implemented in constant rounds. So the following lemma holds.

▶ **Lemma 6.** *Algorithm 1 runs in $O(\Gamma_d \cdot \log \Gamma_p / \epsilon^2)$ rounds. When it terminates, all $r_e = 0$ and all $\rho_S = 0$.*

What remains is to prove its correctness. Let $\boldsymbol{x}^L$ and $\boldsymbol{y}^L$ denote the values of $\boldsymbol{x}$ and $\boldsymbol{y}$ right after the L-th phase. We first prove the feasibility.

▶ **Theorem 7.** *$\boldsymbol{x}^L/f$ and $\boldsymbol{y}^L/((1+\epsilon) \cdot f)$ are feasible solutions to* (3) *and* (4) *respectively.*

Proof. We first show the feasibility of $\boldsymbol{x}^L/f$. Obviously, $\boldsymbol{x}^L/f$ are non-negative. Given any e, whenever we increase x_S by 1 for some $S \ni e$, we divide r_e by a factor $\alpha^{A_{eS}}$. The

[2] The reason behind the choices of parameters is that α should sufficiently close to 1 and f should sufficiently large, such that $\alpha^{\Gamma_d+1} = 1 + O(\epsilon)$ and $\frac{\ln \Gamma_p}{\ln \alpha} \ll f$. See the proof of Theorem 7 for details.

initial value of r_e is 1; and by Lemma 6, finally r_e becomes $\leq \alpha^{-f}$, and then is set to 0. We therefore have

$$\sum_{S \ni e} A_{eS} \cdot x_S^L \geq f,$$

and then conclude the feasibility of $\boldsymbol{x}^L/f$.

In the following, we prove the feasibility of $\boldsymbol{y}^L/f$. Obviously, $\boldsymbol{y}^L/((1+\epsilon)\cdot f)$ are non-negative. What remains is to show that for any S

$$\sum_{e \in S} A_{eS} \cdot y_e^L \leq (1+\epsilon)\cdot f.$$

For the convenience of presentation, define $Y_S := \sum_{e\in S} A_{eS} \cdot y_e$, which only increases during the algorithm's execution. The idea is to upper bound the increment ΔY_S of Y_S in terms of the decrement $\Delta\rho_S$ of ρ_S in each phase.

On the one hand, recalling that the increment of y_e is

$$\Delta y_e = \sum_{S' \in \Xi_\ell(e)} \frac{A_{eS'} \cdot r_e}{\sum_{e\in S'} A_{eS'} \cdot r_e} = \sum_{S' \in \Xi_\ell(e)} \frac{1}{\rho_{S'}} \cdot A_{eS'} \cdot r_e,$$

we have

$$\begin{aligned}\Delta Y_S =&\sum_{e\in S} A_{eS} \cdot \Delta y_e = \sum_{e\in S}\sum_{S' \in \Xi_\ell(e)} \frac{1}{\rho_{S'}} \cdot A_{eS} \cdot A_{eS'} \cdot r_e \leq \sum_{e\in S}\sum_{S' \in \Xi_\ell(e)} \frac{\alpha}{\rho_S} \cdot A_{eS} \cdot A_{eS'} \cdot r_e \\ =&\frac{\alpha}{\rho_S}\sum_{e\in S}\sum_{S' \in \Xi_\ell(e)} A_{eS} \cdot A_{eS'} \cdot r_e.\end{aligned}$$

On the other hand,

$$\begin{aligned}\Delta\rho_S =&\sum_{e\in S} A_{eS}\cdot \Delta r_e = \sum_{e\in S} A_{eS} \cdot r_e \cdot \left(1 - 1/\alpha^{\sum_{S'\in\Xi_\ell(e)} A_{eS'}}\right) \\ &= \sum_{e\in S} A_{eS} \cdot r_e \cdot \frac{1}{\alpha^{\sum_{S'\in\Xi_\ell(e)} A_{eS'}}}\left(\alpha^{\sum_{S'\in\Xi_\ell(e)} A_{eS'}} - 1\right) \\ &\geq \sum_{e\in S} A_{eS} \cdot r_e \cdot \frac{1}{\alpha^{\sum_{S'\in\Xi_\ell(e)} A_{eS'}}}\left(\sum_{S'\in\Xi_\ell(e)} \ln\alpha \cdot A_{eS'}\right) \\ &\geq \frac{\ln\alpha}{\alpha^{\Gamma_d}} \sum_{e\in S}\sum_{S'\in\Xi_\ell(e)} A_{eS}\cdot A_{eS'}\cdot r_e\end{aligned}$$

Combining the two inequalities above, we get

$$\Delta Y_S \leq \frac{\alpha^{\Gamma_d+1}}{\ln\alpha}\cdot\frac{\Delta\rho_S}{\rho_S}.$$

Then, summing this inequality up over all phases, we have

$$\begin{aligned}Y_S^{\text{end}} &= \sum_{\text{each phase}} \Delta Y_S \leq \frac{\alpha^{\Gamma_d+1}}{\ln\alpha}\sum_{\text{each phase}}\frac{\Delta\rho_S}{\rho_S} \leq \frac{\alpha^{\Gamma_d+1}}{\ln\alpha}\int_{\rho_S^{\text{end}}}^{\rho_S^{\text{initial}}} \frac{1}{\rho}\,\mathrm{d}\rho \\ &= \frac{\alpha^{\Gamma_d+1}}{\ln\alpha}\cdot\left(\ln\rho_S^{\text{initial}} - \ln\rho_S^{\text{end}}\right) \leq \frac{\alpha^{\Gamma_d+1}}{\ln\alpha}\cdot(\ln\alpha\cdot f + \ln\Gamma_p) \\ &= \alpha^{\Gamma_d+1} f + \frac{\alpha^{\Gamma_d+1}\ln\Gamma_p}{\ln\alpha} \leq (1+\epsilon) f.\end{aligned}$$

◀

By Proposition 5 (a), we have $\sum_S x_S^L = \sum_e y_e^L$, which means

$$\sum_e y_e^L / ((1+\epsilon) \cdot f) \leq (1+\epsilon) \sum_S x_S^L / f.$$

We therefore conclude that $\boldsymbol{x}^L/f$ and $\boldsymbol{y}^L/((1+\epsilon)\cdot f)$ are $(1+\epsilon)$-approximate solutions to (3) and (4) respectively. By putting it and Lemma 6 together, we finish the proof of Theorem 3.

3 Conclusion

This paper proposes a simple $(1+\epsilon)$-approximation CONGEST algorithm for row-sparse fractional covering problems and column-sparse fractional packing problems. It runs in $O\left(A_{\max} \cdot \log \Gamma_p / \epsilon^2\right)$ rounds, where $\Gamma_p = \max_j \sum_i A_{ij}$ and $\Gamma_d = \max_i \sum_j A_{ij}$. Our algorithm is simpler than the algorithm of [8], worsens the A_{max}-dependency, but improves the ϵ-dependency. For future work, it is an intriguing open problem, proposed by Suomela [13], whether constant round, constant-factor approximation CONGEST algorithms exist for row-sparse, column-sparse FCP/FPP instances – a special kind of RS-FCP/CS-FPP where the number of nonzero entries in each column of A is also bounded. Our algorithm and the algorithm of [8] are both such algorithms for instances where $A_{\max}$ is bounded.

References

1 Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. Distributed approximate maximum matching in the CONGEST model. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.DISC.2018.6.

2 Baruch Awerbuch and Rohit Khandekar. Stateless distributed gradient descent for positive linear programs. *SIAM J. Comput.*, 38(6):2468–2486, 2009. doi:10.1137/080717651.

3 Yair Bartal, John W. Byers, and Danny Raz. Global optimization using local information with applications to flow control. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 303–312. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646119.

4 Friedrich Eisenbrand, Stefan Funke, Naveen Garg, and Jochen Könemann. A combinatorial algorithm for computing a maximum independent set in a t-perfect graph. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 517–522. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644194.

5 Fabian Kuhn. *The price of locality: exploring the complexity of distributed coordination primitives*. PhD thesis, ETH Zurich, 2005. URL: https://d-nb.info/977273725.

6 Fabian Kuhn. Local approximation of covering and packing problems. In *Encyclopedia of Algorithms*, pages 1129–1132. Springer New York, 2016. doi:10.1007/978-1-4939-2864-4_209.

7 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In Soma Chaudhuri and Shay Kutten, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 300–309. ACM, 2004. doi:10.1145/1011767.1011811.

8 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 980–989. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109666.

9 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, 2016. doi:10.1145/2742012.

10 Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Comput.*, 17(4):303–310, 2005. doi:10.1007/S00446-004-0112-5.

11 Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2010. doi:10.1007/978-3-642-15763-9_48.

12 Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 121–129. ACM, 1993. doi:10.1145/167088.167127.

13 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. doi:10.1145/2431211.2431223.

Uniform Polynomial Kernel for Deletion to $K_{2,p}$ Minor-Free Graphs

William Lochet
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Roohani Sharma
Department of Informatics, University of Bergen, Norway

Abstract

In the $\mathcal{F}$-Deletion problem, where $\mathcal{F}$ is a fixed finite family of graphs, the input is a graph G and an integer k, and the goal is to determine if there exists a set of at most k vertices whose deletion results in a graph that does not contain any graph of $\mathcal{F}$ as a minor. The $\mathcal{F}$-Deletion problem encapsulates a large class of natural and interesting graph problems like Vertex Cover, Feedback Vertex Set, Treewidth-η Deletion, Treedepth-η Deletion, Pathwidth-η Deletion, Outerplanar Deletion, Vertex Planarization and many more. We study the $\mathcal{F}$-Deletion problem from the kernelization perspective. In a seminal work, Fomin et al. [FOCS 2012] gave a polynomial kernel for this problem when the family $\mathcal{F}$ contains at least one planar graph. The asymptotic growth of the size of the kernel is not uniform with respect to the family $\mathcal{F}$: that is, the size of the kernel is $k^{f(\mathcal{F})}$, for some function f that depends only on $\mathcal{F}$. Later Giannopoulou et al. [TALG 2017] showed that the non-uniformity in the kernel size bound is unavoidable as Treewidth-η Deletion cannot admit a kernel of size $\mathcal{O}(k^{\frac{\eta+1}{2}-\epsilon})$, for any $\epsilon > 0$, unless NP $\subseteq$ coNP/poly. On the other hand it was also shown that Treedepth-η Deletion admits a uniform kernel of size $f(\mathcal{F}) \cdot k^6$ depicting that there are subclasses of $\mathcal{F}$ where the asymptotic kernel sizes do not grow as a function of the family $\mathcal{F}$. This work led to the question of determining classes of $\mathcal{F}$ where the problem admits uniform polynomial kernels.

In this paper, we show that if all the graphs in $\mathcal{F}$ are connected and $\mathcal{F}$ contains $K_{2,p}$ (a bipartite graph with 2 vertices on one side and p vertices on the other), then the problem admits a uniform kernel of size $f(\mathcal{F}) \cdot k^{10}$. The graph $K_{2,p}$ is one natural extension of the graph θ_p, where θ_p is a graph on two vertices and p parallel edges. The case when $\mathcal{F}$ contains θ_p has been studied earlier and serves as (the only) other example where the problem admits a uniform polynomial kernel.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Uniform polynomial kernel, $\mathcal{F}$-minor-free deletion, complete bipartite minor-free graphs, $K_{2,p}$, protrusions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.46

1 Introduction

For any fixed finite family of (multi-)graphs $\mathcal{F}$, in the $\mathcal{F}$-Deletion problem, given as input a graph G and a positive integer k, the task is to determine whether the deletion of a set of at most k vertices results in a graph that does not contain any graph of $\mathcal{F}$ as a minor. The $\mathcal{F}$-Deletion problem encompasses various natural and interesting problems such as Vertex Cover, Feedback Vertex Set, Treewidth-η Deletion, Treedepth-η Deletion, Pathwidth-η Deletion, Outerplanar Deletion, Vertex Planarization and much more. As a result of the seminal work of Lewis and Yannakakis [18] the problem is known to be NP-complete. By a celebrated result of Robertson and Seymour [22] every $\mathcal{F}$-Deletion problem is non-uniformly FPT, that is, for every integer k, there exists an algorithm that solves the problem in $f(k) \cdot n^3$ time, where n is the number of vertices in the input graph. However, when the family $\mathcal{F}$ is given explicitly, the problem is uniformly FPT because the excluded minors for the graphs that are YES instances of the problem can be computed

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 46; pp. 46:1–46:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

explicitly from the result by Adler et al. [1]. Another breakthrough result by Fomin et al. [12] shows that the problem admits an algorithm with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ when all the graphs in $\mathcal{F}$ are connected and $\mathcal{F}$ contains a planar graph. The class considered by Fomin et al. seems a little restrictive at first, but it already encapsulates the classical problems mentioned above except for Vertex Planarization. In fact, the class of problems considered by Fomin et al. [12] are essentially about deleting k vertices to get to a graph of constant treewidth, since graphs that exclude a planar graph H as a minor have treewidth at most $|V(H)|^{\mathcal{O}(1)}$ [3].

One of the major highlights of the result by Fomin et al. [12] is also a polynomial (in k) kernel for the $\mathcal{F}$-Deletion problem when $\mathcal{F}$ contains a planar graph. A noteworthy feature of their kernelization algorithm is that the size of the kernel is $f(\mathcal{F}) \cdot k^{g(\mathcal{F})}$, for some functions f, g that depend only on the family $\mathcal{F}$. In particular, the exponent of the size of the kernel depends on the family $\mathcal{F}$. Such kernels are called *non-uniform* kernels as the asymptotic size of the kernel varies with the family $\mathcal{F}$. Such a result opens up questions about the existence of a kernel of size $f(\mathcal{F}) \cdot k^{\mathcal{O}(1)}$ for $\mathcal{F}$-Deletion. Such a kernel is called a *uniform polynomial kernel.*

Soon after the result of Fomin et al., Giannopoulou et al. [14] showed that the size of the polynomial kernel by Fomin et al. is essentially tight in the sense that $\mathcal{F}$-Deletion cannot admit a kernel of size $f(\mathcal{F}) \cdot k^{\mathcal{O}(1)}$, under reasonable complexity assumptions. In particular, they showed that Treewidth-η Deletion, a special case of $\mathcal{F}$-Deletion where $\mathcal{F}$ contains a planar graph, cannot admit a uniform polynomial kernel unless NP $\subseteq$ coNP/poly, even when the parameter is the vertex cover of the graph. More specifically they showed that Treewidth-η Deletion cannot admit a kernel on $\mathcal{O}(x^{\frac{\eta+1}{2}-\epsilon})$ vertices, for any $\epsilon > 0$, where x is the size of the vertex cover in the input graph. They also complemented this result by showing that Treedepth-η Deletion, another special case of $\mathcal{F}$-Deletion when $\mathcal{F}$ contains a path, admits a (uniform) kernel of size $f(\eta) \cdot k^6$ for some function f.

Other than Treedepth-η Deletion, the only other family $\mathcal{F}$ for which $\mathcal{F}$-Deletion is known to admit a uniform polynomial kernel (of size $f(\mathcal{F}) \cdot k^2 \log^{3/2} k)$) is when the graph $\theta_p \in \mathcal{F}$ [11]. Here θ_p is a graph with two distinct vertices and p parallel edges between them.

This contrast in the behaviour of the asymptotic size of polynomial kernels, obtained for different specializations of $\mathcal{F}$-Deletion, leads to the question- *under what restrictions of $\mathcal{F}$, does the $\mathcal{F}$-Deletion problem admit uniform kernels?* Our study investigates this direction and exhibits an infinite collection of families $\mathcal{F}$ for which the $\mathcal{F}$-Deletion problem admits a uniform polynomial kernel.

Our Result. We show that $\mathcal{F}$-Deletion admits a kernel of size $f(\mathcal{F}) \cdot k^{10}$, when all the graphs in $\mathcal{F}$ are connected and $K_{2,p} \in \mathcal{F}$, where $K_{2,p}$ is a complete bipartite graph on 2 vertices on one side and p vertices on the other. Henceforth, for any positive integer p, let $\mathcal{F}_p$ denote an arbitrary finite family of connected graphs such that $K_{2,p} \in \mathcal{F}_p$. The $\mathcal{F}_p$-Deletion problem is formally defined as follows: given a graph G and an integer k, does there exist $X \subseteq V(G)$, $|X| \leq k$ such that $G - X$ has no graph of $\mathcal{F}_p$ as a minor?

In the remaining paper we subsume the factors depending on $\mathcal{F}_p$ in the $\mathcal{O}(\cdot)$ notation. Also, a polynomial running time refers to a running time that is polynomial in the input size where the exponent of the polynomial is an absolute constant (and hence does not depend on $\mathcal{F}_p$ or k). Thus, our kernelization algorithm runs in "purely" polynomial time.

▶ **Theorem 1.1.** *$\mathcal{F}_p$-Deletion admits a kernel of size $\mathcal{O}(k^{10})$.*

$K_{2,p}$-free graphs. The class $\mathcal{F}_p$-Deletion that we consider is, *and has to be*, more restrictive than what is considered by Fomin et al. [12], given the hardness result by Giannopoulou et al. [14]. In the following points we motivate our interests in the study of the chosen family $\mathcal{F}_p$.

(1) Generalizes outerplanarity. A graph is called *outerplanar* if there exists a planar embedding of it where all the vertices lie on the outer face. In the Outerplanar Deletion problem, given as input a graph G and an integer k the goal is to decide if the deletion of at most k vertices results in an outerplanar graph. A well-known consequence of Wagner's characterization of planar graphs implies that the Outerplanar Deletion problem is equivalent to the $\mathcal{F}$-Deletion problem where $\mathcal{F} = \{K_{2,3}, K_4\}$, where K_4 is a complete graph on 4 vertices. Clearly, $\mathcal{F}_p$-Deletion encapsulates and generalizes the Outerplanar Deletion problem.

(2) Challenge in extension from θ_p. As mentioned earlier, prior to the polynomial kernelization result of Fomin et al. [12] for general families $\mathcal{F}$ containing some planar graph, Fomin et al. [11] gave a uniform polynomial kernel for $\mathcal{F}$-Deletion when $\theta_p \in \mathcal{F}$. Such families already encapsulate classical problems like Vertex Cover, Feedback Vertex Set and Diamond Hitting [11].

Observe that $K_{2,p}$ is a natural extension of θ_p as it can be obtained from θ_p by subdividing each of its edges once. This seemingly simple extension of θ_p already poses great technical challenges, thereby disallowing to lift the kernelization techniques used in the θ_p case to the $K_{2,p}$ case. As we describe in detail later (see Section 3), the challenge in making a uniform polynomial kernel for special cases of $\mathcal{F}$-Deletion lie in what we call the *degree reduction phase* of [12]. We elaborate on this later but let us give some overview of it already here. Let S be some approximate solution to the problem of size $k^{\mathcal{O}(1)}$. Let C be some connected component of $G - S$. If one can bound the degree of each vertex of $x \in S$ in the set C by $f(\mathcal{F}) \cdot k^{\mathcal{O}(1)}$ (where the degree of k is independent of $\mathcal{F}$), then following the approach of [12], one can get a *uniform* polynomial kernel for the $\mathcal{F}$-Deletion problem.

Using the above, a uniform polynomial kernel for $\mathcal{F}$-Deletion when $\theta_p \in \mathcal{F}$ follows very easily: let S be a 1-redundant solution to the problem of size $\mathcal{O}(k^2)$, that is for each $x \in S$, $S \setminus x$ is a solution to the problem. As we will see later such sets of $\mathcal{O}(k^2)$ size can be found easily. Let C be a connected component of $G - S$. Then for any $x \in S$, the degree of x in C is at most $p - 1$, as otherwise there would be θ_p as a minor in $G[C \cup \{x\}]$, contradicting that $S \setminus \{x\}$ is a solution. Thus, in this case one can in fact, bound the degree of x in C by $\mathcal{O}(1)$.

The above simple argument for bounding the degree of x fails completely when the forbidden minor is a subdivided θ_p, that is a $K_{2,p}$. For example, consider a graph containing $n + 1$ vertices, where one vertex is adjacent to all the other n vertices and these n vertices are connected to form a path. This graph has θ_n as a minor but no $K_{2,3}$ as a minor.

(3) Interesting structural graph properties of $K_{2,p}$-free graphs. From a graph theoretic viewpoint, excluding certain classes of graphs as minors seem to give close connections to some interesting graph properties. One of the most interesting conjectures at present demonstrating this is the Hadwiger's Conjecture which states that the chromatic number of any graph that avoids K_t as a minor is at most $t - 1$. Following this line of work, graph theorists have developed a special interest in the class of graphs that exclude a *complete bipartite graph* as a minor [10, 7, 23, 5, 20, 6, 9]. Together with connectivity requirements, and possibly other assumptions, graphs with no $K_{q,p}$ as a minor can be shown to have interesting properties relating to toughness, hamiltonicity, and other traversability properties [4, 5, 21]. Particular attention has been given to the case when $q = 2$, as most of these properties appear to hold for this special case too. Note that any graph avoiding $K_{2,p}$ as a minor is at

most $p-1$-connected. Results in the literature show that 3-connected $K_{2,4}$-free graphs are Hamiltonian and the 2-connected $K_{2,4}$-free graphs have a Hamiltonian Path [10]. Also [7] shows that every 2-connected $K_{2,4}$-free graph contains two vertices whose deletion results in an outerplanar graph. Graphs on n vertices that are $K_{2,p}$-free are known to have long cycles, in particular, cycles of length at least n/p^{p-1} [5]. Another result shows that the number of edges in an n-vertex $K_{2,p}$-free graph, for $p \geq 2$, is at most $(1/2)(p+1)(n-1)$ [6]. This literature thus suggests that the class of $K_{2,p}$-free graphs exhibit interesting graph theoretic properties and hence, it could be worth to study this graph class algorithmically.

(4) Extremal limit before encapsulating planarization. In continuation of the above point, at the front of avoiding a $K_{q,p}$ as a minor, it must be noted that the case when $q = 3$ already encompasses the classical and notorious Vertex Planarization problem. In this problem, the goal is to delete at most k vertices such that the resulting graph is a planar graph. This problem is equivalent to $\mathcal{F}$-Deletion when $\mathcal{F} = \{K_{3,3}, K_5\}$ because of Wagner's characterization of planar graphs. Note that none of the graphs in $\mathcal{F}$ are planar. The first constructive FPT algorithm for this problem was given by Marx and Schlotter in 2007 [19], which was followed by an improved algorithm by Kawarabayashi [17]. This was followed by the current best known algorithm for the problem by Jansen et al. [16] in 2014 that runs in time $2^{\mathcal{O}(k \log k)} \cdot n$. Over this long period of improvements, one important open question that has intrigued the community is the question about the existence of a polynomial kernel for the problem. Recently Jansen and Wlodarczyk [15] gave a lossy polynomial kernel for this problem. But it seems for now that getting a (non-lossy) polynomial kernel for this problem may require more novel ideas. Thus, on the front of avoiding complete bipartite minors, avoiding anything beyond $K_{2,p}$ must first confront the Vertex Planarization problem.

Roadmap. In Section 2 we define basic notations and definitions. In Section 3 we describe all the (five) steps of our kernelization algorithm. In particular, we state formally the five main lemmas that we prove to give the complete proof of Theorem 1.1. In Section 4 we focus on our main technical contribution of this work which is what we call the degree reduction phase of the kernelization algorithm (step 2 of the 5 steps). We give an overview of the key ideas of this phase, followed by formal proofs. In Section 5 we conclude with some open questions. The details of the 4 other phases of the algorithm that are described in Section 3 have been omitted because of space constraints. Also the proofs of lemmas marked with $\star$ have been omitted due to space constraints.

2 Preliminaries

For standard notations and terminology that is not defined here, we refer to [8]. For the definition of kernelization and related terminology we refer to the book [13]. Throughout the paper, $h = \max_{H \in \mathcal{F}_p} |V(H)|$.

General. For positive integers $i < j$, $[i]$ denotes the set $\{1, \ldots, i\}$ and $[i, j]$ denote the set $\{i, i+1, \ldots, j\}$. Given a sequence, an *interval* is a set of consecutive entries of the sequence. The *length of the interval* is the number of entries in it.

Graphs. For $u, v \in V(G)$ a (u, v)-path in G is a path from u to v. The *internal* vertices of a path are the vertices of the path that are not its end-points. For $X \subseteq V(G)$, a path is called *X-free* if none of the vertices of X appear as internal vertices of the path. For $X, Y \subseteq V(G)$, $Z \subseteq V(G)$ is called an *(X, Y)-cut* if $G - Z$ has no path from a vertex of X to a vertex of Y. When X or Y are singletons we drop the braces around them in this notation.

Boundaried graphs and boundaried minors. For any positive integer t, a *t-boundaried graph* is a graph G together with a specially assigned vertex set of size at most t called the boundary of G. Also each vertex in the boundary is labelled with a distinct integer from $[t]$. A t-boundaried graph H is a *minor* of a t-boundaried graph G if H can be obtained from G by deleting vertices or edges or contracting edges, but never contracting edges with both endpoints being the vertices in the boundary. If we contract an edge between a boundary vertex u and a non-boundary vertex v, the resulting vertex is a boundary vertex with the same label as that of u. For a t-boundaried graph G, for any positive integer δ, the *δ-folio*(G) is the set of all t-boundaried graphs of size at most δ that can be obtained as boundaried minors in G.

3 The steps of the kernelization algorithm

The kernelization algorithm of Theorem 1.1 has five phases which we term as (1) redundant solution, (2) degree reduction, (3) component reduction, (4) protrusion decomposition, and (5) protrusion replacement. We emphasize here that the degree reduction phase is our main technical contribution of this work and our special choice of starting with a redundant solution in phase one is key for the second phase. Other than this, the overall structure of our kernelization algorithm follows the footprints of that of [12]. For any graph G, let $\texttt{cc}(G)$ denote the set of connected components of G. Throughout the presentation, $h := \max_{F \in \mathcal{F}_p} |V(F)|$. Let (G, k) be an instance of $\mathcal{F}_p$-DELETION problem.

Redundant solution. In the *first* phase, we find a 1-redundant solution S in G of size $\mathcal{O}(k^2)$. A 1-*redundant solution* in G is a set of vertices S such that for every $x \in S$, $S \setminus \{x\}$ is a solution of $\mathcal{F}_p$-DELETION. This step is different from that of [12] where any solution modulator (for example, any approximate solution) works. Formally, we prove the following lemma.

▶ **Lemma 3.1** ($\star$, Redundant solution). *Given an instance (G, k) of $\mathcal{F}_p$-DELETION, there is a polynomial-time algorithm that either outputs $x \in V(G)$ such that $(G - \{x\}, k - 1)$ is an equivalent instance of (G, k), or outputs a 1-redundant solution of (G, k) of size $\mathcal{O}(k^2)$, or concludes correctly that (G, k) is a* NO *instance of $\mathcal{F}_p$-DELETION.*

If Lemma 3.1 outputs $x \in V(G)$ then we apply a reduction rule and output the instance $(G - \{x\}, k - 1)$ as an equivalent instance. If it outputs NO, then we output a trivial constant sized NO instance of $\mathcal{F}_p$-DELETION. Otherwise, we have a 1-redundant solution of size $\mathcal{O}(k^2)$, that we denote by S in all the subsequent phases.

Degree reduction. The input to this phase is the instance (G, k) together with a 1-redundant solution S. The goal is to design a reduction rule that bounds the size of the set of neighbours of x in C, for each $x \in S$ and $C \in \texttt{cc}(G - S)$, by $k^{\mathcal{O}(1)}$. An edge uv in G is called *irrelevant* if the instance (G, k) is equivalent to the instance $(G - uv, k)$.

Let a denote the number of 2-boundaried graphs on at most $2h+6$ vertices. Let $g : [a] \to \mathbb{N}$ be a function such that $g(1) := 18 \cdot (2h + 6)$ and for each $i > 1$, $g(i) := 18 \cdot (2h + 6) \cdot g(i - 1)$. Also let $\texttt{degree-bound} := p^3 \cdot (|S| \cdot (p + k + 1) + 1) \cdot g(a)$. We prove the following lemma.

▶ **Lemma 3.2** (Irrelevant edge). *Let (G, k) be an instance of $\mathcal{F}_p$-DELETION and S be a 1-redundant solution in G. Let $x \in S$ and $C \in \texttt{cc}(G - S)$. If $|N(x) \cap C| \geq \texttt{degree-bound}$, then there exists $u \in N(x) \cap C$, such that xu is irrelevant. Moreover, such a vertex u can be found in polynomial time.*

The key insights and the proof of Lemma 3.2 are delegated to Section 4. We use Lemma 3.2 as long as there exists $x \in S$ and $C \in \texttt{cc}(G-S)$ such that $|N(x) \cap C| \geq \texttt{degree-bound}$. If u is the vertex reported by Lemma 3.2, then we apply a reduction rule and output an equivalent instance $(G-uv, k)$. Therefore at the end of this stage we can assume that for each $x \in S$ and $C \in \texttt{cc}(G-S)$, $|N(x) \cap C| < \texttt{degree-bound} = \mathcal{O}(k^3)$. It is crucial to note here that the exponent of k in $\texttt{degree-bound}$ is independent of $\mathcal{F}$.

Component reduction. In this phase, we design a reduction rule whose exhaustive application guarantees that $|\texttt{cc}(G-S)| = \mathcal{O}(|S|^2 \cdot k) = \mathcal{O}(k^5)$. Formally we prove the following lemma. Let $\texttt{comp-bound}_h := (10+p) \cdot (h+1)^2 \cdot 2^{\binom{h+1}{2}}$. Note that $\texttt{comp-bound}_h$ depends only on h, p (and therefore only on $\mathcal{F}_p$).

▶ **Lemma 3.3** (⋆, Component reduction). *Let (G,k) be an instance of $\mathcal{F}_p$-Deletion and S be some solution of $\mathcal{F}_p$-Deletion (not necessarily optimal). If $|\texttt{cc}(G-S)| \geq \texttt{comp-bound}_h \cdot |S|^2 \cdot k$ [1], then there exists $C \in \texttt{cc}(G-S)$ such that the instance (G,k) is equivalent to $(G-C,k)$. Moreover, such a component C can be found in polynomial time.*

Apply Lemma 3.3 on the instance (G,k) and the set S from phase 1 until $|\texttt{cc}(G-S)| = \mathcal{O}(k^5)$.

Protrusion decomposition. Note that at the end of phase three, we can bound the size of the set of neighbours of S ($|N(S)|$) by $\mathcal{O}(k^{10})$: indeed from phase three (Lemma 3.3) $|\texttt{cc}(G-S)| = \mathcal{O}(k^5)$, from phase 2, for each $x \in S$, $|N(x) \cap C| = \mathcal{O}(k^3)$ and $|S| = \mathcal{O}(k^2)$. We use this to obtain a protrusion decomposition of G with $\mathcal{O}(k^{10})$ protrusions. This is defined below.

For a graph G, let $\texttt{tw}(G)$ denote the treewidth (see [8] for the definition) of G. For a positive integer a, an *a-protrusion* in G is a set of vertices $X \subseteq V(G)$ such that $\texttt{tw}(G[X]) \leq a$ and $|N_G(X)| \leq a$. For positive integers a, b, c, an *(a,b,c)-protrusion decomposition* of G is a partition of $V(G) = V_0 \uplus V_1 \uplus \ldots \uplus V_r$ such that the following holds: (1) $|V_0| \leq a$, (2) $r \leq b$, (3) for each $i \in [c]$, $N(V_i) \subseteq V_0$, and, (4) for each $i \in [c]$, V_i is a c-protrusion in G.

In this phase, we prove the following lemma.

▶ **Lemma 3.4** (⋆, Protrusion decomposition). *Let G be a graph and $S \subseteq V(D)$ such that $tw(G-S) \leq \eta$ and $G-S$ has at most ζ connected components. If $|N(S) \cap C| \leq \alpha$ for every connected component of $G-S$, then G admits a $(|S|+2\alpha\eta\zeta, 6\alpha\zeta, 2\eta)$-protrusion decomposition and can be computed in polynomial time.*

We use Lemma 3.4 on G and the set S from phase 1. Since S is a solution to $\mathcal{F}_p$-Deletion, $K_{2,p} \in \mathcal{F}_p$ and $K_{2,p}$ is planar, $\texttt{tw}(G-S) = \mathcal{O}((p+2)^9)$ [3]. Thus, in Lemma 3.4, on input G, S, $\eta = \mathcal{O}((p+2)^9)$, from phase 3 $\zeta = \mathcal{O}(k^5)$ and, from phase 2 $\alpha = \mathcal{O}(k^3) \cdot |S| = \mathcal{O}(k^5)$. Thus, we get an $(\mathcal{O}(k^{10}), \mathcal{O}(k^{10}), \mathcal{O}(1))$-protrusion decomposition of G.

Protrusion replacement. Let $V_0 \uplus V_1 \uplus \ldots V_r$ be the $(\mathcal{O}(k^{10}), \mathcal{O}(k^{10}), \mathcal{O}(1))$-protrusion decomposition of G obtained from the previous phase. Note that in order to bound the size of the whole graph G, it remains to bound the size of protrusion V_i for each $i \in \{1, \ldots, r\}$. This is done in this final phase. To reduce the size of the protrusions, we use the fact that the $\mathcal{F}_p$-Deletion problem has Finite Integer Index. This allows one to "replace" each protrusion V_i with a vertex set whose size depends only on $\mathcal{F}_p$.

[1] The bound on $\texttt{comp-bound}_h$ has not been optimized.

Recall η, ζ, α from Lemma 3.4. The following proposition from [2, 12] implies a reduction rule if the size of any V_i, $i \in \{1, \ldots, r\}$, is larger than a fixed constant q that depends only on $\mathcal{F}_p$.

▶ **Proposition 3.5** (Protrusion replacer, [2, 12]). *Let (G, k) be an instance of $\mathcal{F}_p$-Deletion and let $V_i \subseteq V(G)$ be a 2η-protrusion in G of size at least q, where q is a fixed constant that depends only on $\mathcal{F}_p$. Then there exists a polynomial-time algorithm that outputs an equivalent instance (G', k') such that $|V(G')| < |V(G)|$ and $k' \leq k$.*

Finally this implies the following corollary. Again recall η, ζ, α from Lemma 3.4 and q from Proposition 3.5.

▶ **Corollary 3.6.** *If $|V(G)| \geq |S| + 2\alpha\eta\zeta + 6q\alpha\zeta k$, then the reduction rule implied by Proposition 3.5 is applicable.*

This implies that when the reduction rule of Lemma 3.5 is not applicable, $|V(G)| = \mathcal{O}(k^{10})$. The proof of Theorem 1.1 follows from Lemmas 3.1,3.2,3.3,3.4 and Corollary 3.6.

4 The degree reduction phase

In this section we elaborate on the degree reduction phase that we described in Section 3. Recall that (G, k) is an instance of $\mathcal{F}_p$-Deletion and $S \subseteq V(G)$ is a 1-redundant solution, that is for each $x \in S$, $S \setminus \{x\}$ is a solution. Fix $x \in S$ and $C \in \texttt{cc}(G - S)$ such that $|N(x) \cap C| \geq \texttt{degree-bound}$. Because S is 1-redundant, $G[C \cup \{x\}]$ is $K_{2,p}$-free. Set $X := N(x) \cap C$. The goal is to design a reduction rule that bounds the size of X by $k^{\mathcal{O}(1)}$.

4.1 Overview and key insights

We give an overview of the key insights in the proof of Lemma 3.2. The degree reduction phase has two main steps. In this first step we find a subset of C that has a very nice structure and in the second step we exploit this structure to "re-build" minor models of small graphs so that they avoid an irrelevant edge.

A nicely structured set $C^\sigma \subseteq C$. As a first step, we exploit the fact that $G[C \cup \{x\}]$ is $K_{2,p}$-free to obtain a subset $C^\sigma \subseteq C$ containing $g(a)$ (recall a, g from Section 3) neighbours of x such that C^σ has a very nice "chain-like" structure. The definition of the structure is formalized in Definition 4.1 as an *x-good sequence.* Also see Figure 1. The proof of its existence is given in Lemma 4.3. Below we state informally the nice structure of C^σ that we achieve.

The chain structure: The set C^σ contains $g(a)$ vertices of X, say ordered $(u_1, \ldots, u_{g(a)})$. For each other vertex $v \in C^\sigma$, v is on some X-free (u_i, u_{i+1})-walk in C. For each $i \in [g(a) - 1]$, let V_i^σ be the set of vertices on some X-free (u_i, u_{i+1})-walk in C. Then $V_i^\sigma \neq \emptyset$. For each $i, j \in [g(a) - 1]$, $i \neq j$, $V_i^\sigma \cap V_j^\sigma = \emptyset$. The set $\{u_i, u_{i+1}\} \cup V_i^\sigma$ is called a *block* of C^σ.

The boundary to C: The vertices $u_1, u_{g(a)}$ are the boundary vertices of C^σ in C. That is, no vertex of $C^\sigma \setminus \{u_1, u_r\}$ has any neighbour in $C \setminus C^\sigma$.

Neighbours in S after removing a solution: Lastly, for any $\mathcal{F}_p$-Deletion solution T of size at most $k + 1$, in $G - T$, the $N(C^\sigma) \cap S \subseteq \{x\}$.

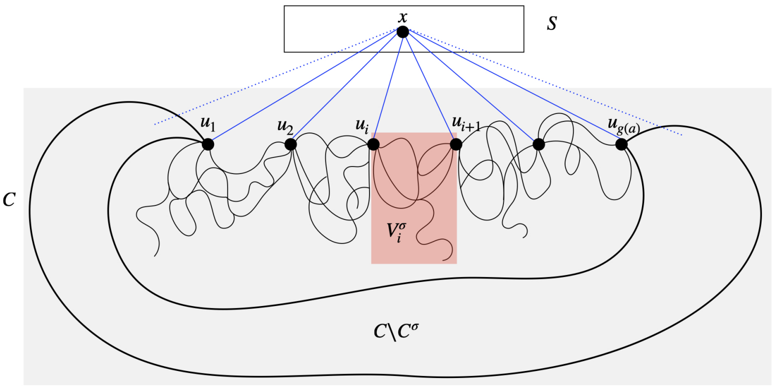


Figure 1 The structure of C^σ: The part with the grey background is a connected component C of $G - S$. The blue edges connect x to its neighbours in C.

The structure of minors of small graphs in $\mathcal{F}_p$ restricted to C^σ. In the second step, we exploit this structure of C^σ to find a vertex $u \in C^\sigma \cap X$ such that xu is an irrelevant edge. To show that xu is irrelevant, we want to show that if there is a set of at most k vertices T in $G - xu$ such that $(G - xu) - T$ is $\mathcal{F}_p$-free, then $G - T$ is also $\mathcal{F}_p$-free. In particular we will show that if $G - T$ has *any* graph H on at most h vertices as a minor, and there is a minor model of H in $G - T$ that uses the edge xu, then there is also a minor model of H in $G - T$ that does not use the edge xu (Lemmas 4.14 and 4.16).

For doing the above, the key insight is to focus on the minor model of H restricted to $C^\sigma \cup \{x\}$.

4.2 Proof of Lemma 3.2 (Irrelevant edge)

The goal of this section is to prove Lemma 3.2.

▶ **Lemma 3.2** (Irrelevant edge). *Let (G, k) be an instance of $\mathcal{F}_p$-Deletion and S be a 1-redundant solution in G. Let $x \in S$ and $C \in \mathtt{cc}(G - S)$. If $|N(x) \cap C| \geq \mathtt{degree\text{-}bound}$, then there exists $u \in N(x) \cap C$, such that xu is irrelevant. Moreover, such a vertex u can be found in polynomial time.*

An *x-sequence* is a sequence σ of a subset of vertices of X. If $\sigma = (u_1, \ldots, u_r)$ is an x-sequence, for each $i \in [r-1]$, the set V_i^σ contains each vertex that appears on some X-free (u_i, u_{i+1})-walk in C. An *i-block* of σ refers to the set $\{u_i, u_{i+1}\} \cup V_i^\sigma$. An *$r$-block* is simply the vertex u_r. By a *block* of σ we simply refer to some i-block of σ. By the *endpoints* of an i-block, we refer to the vertices u_i and u_{i+1}. Further $C^\sigma := \bigcup_{i=1}^{r-1} V_i^\sigma \cup \bigcup_{i=1}^{r} \{u_i\}$. The *length* of the x-good sequence σ is r.

▶ **Definition 4.1** (x-good sequence). *An x-sequence $\sigma = (u_1, \ldots, u_r)$ is called an x-good sequence if the following holds.*

1. *For every $i \in [r-1]$, there is an X-free (u_i, u_{i+1})-walk in C.*
2. *For every $i \in [2, r-1]$, $\{u_{i-1}, u_{i+1}\}$ is a (u_i, X)-cut.*
3. *For any $\mathcal{F}_p$-Deletion set T in G of size at most $k+1$, if $x \notin T$, $N(C^\sigma) \cap (S \setminus T) = \{x\}$.*
4. *No vertex of $C^\sigma \setminus \{u_1, u_r\}$ has a neighbour in $C \setminus C^\sigma$.*

▶ **Lemma 4.2** (⋆). *If $\sigma = (u_1, \dots, u_r)$ is an x-good sequence then, for any minimal $\mathcal{F}_p$-*Deletion *solution T in G of size at most $k+1$ that does not contain x, $\{x\} \subseteq N(C^\sigma \setminus \{u_1, u_r\}) \setminus T \subseteq \{x, u_1, u_r\}$. Also, there exists a minimum $\mathcal{F}_p$-*Deletion *solution T^* in G such that $|T^* \cap C^\sigma| \leq 3$.*

In Section 4.2.1 we show that if X is large then there is a large x-good sequence (Lemma 4.3). In Section 4.2.2 we show how to find an irrelevant edge xu, where $u \in \sigma$, given a large x-good sequence σ.

4.2.1 Finding a large x-good sequence

The goal of this section is to prove the following lemma.

▶ **Lemma 4.3.** *If $|X| \geq p^3 \cdot (|S| \cdot (p+k+1)+1) \cdot g(a)$, then there exists an x-good sequence of length $g(a)$. In fact, such a sequence can be found in polynomial time.*

We start by taking some natural steps towards the construction of some x-sequence that has Property 1 of Definition 4.1. We later perform more steps towards proving other properties of Definition 4.1. Let us define an ordered partition of $X = (L_0, \dots, L_l)$ inductively as follows. Fix an arbitrary vertex $x_0 \in X$ and let L_0 be $\{x_0\}$. Now suppose $L_0, \dots, L_i$ are defined, we define L_{i+1} as the set of vertices of $X \setminus \{\bigcup_{j \in [0..i]} L_j\}$ that are reachable from L_i by X-free paths in $G[C]$. We next prove a series of claims about this partition $X = (L_0, \dots, L_l)$. Observe that each vertex of C is on some X-free (L_i, L_{i+1})-walk.

▶ **Lemma 4.4** (⋆). *For every $i \in [0, l]$, $|L_i| \leq p - 1$.*

Observe that, if $|X| \geq p^3 \cdot (|S| \cdot (p+k+1)+1) \cdot g(a)$, then $l \geq p^2(|S| \cdot (p+k+1)+1)g(a)$.

The next definition and the upcoming steps help to ensure Property 2 of Definition 4.1 (this is formally proved in Lemma 4.11). For every $i \in [0, l-1]$, we say that a vertex $v \in L_i$ is *dangerous* if the set of vertices in X that are reachable from $L_i \setminus v$ by paths whose internal vertex set is disjoint from $(X \setminus L_{i+1})$, is exactly L_{i+1}. Observe that for any vertex $v \in L_i$ which is not dangerous, there is a vertex v' in L_{i+1} such that v' is not reachable from $L_i \setminus v$ by an $(X \setminus L_{i+1})$-free path. Such a vertex v' is called a *witness of a non dangerous vertex v*.

▶ **Lemma 4.5** (⋆). *The number of indices $i \in [0, l]$ such that L_i contains a dangerous vertex is at most $p - 1$.*

▶ **Lemma 4.6** (⋆). *For every $i \in [0, l-1]$, if no vertex of L_i is dangerous, then $|L_i| \leq |L_{i+1}|$.*

▶ **Lemma 4.7** (⋆). *Let $t = (|S| \cdot (p+k+1)+1)g(a)$. There exists $i \in [l-t]$ such that none of $L_i, \dots, L_{i+t}$ contains dangerous vertices and $|L_i| = |L_{i+s}|$ for all $s \in [t]$.*

Without loss of generality, let $L_1, \dots, L_t$ denote the interval of $(L_0, \dots, L_l)$ from Lemma 4.7 that do not contain a dangerous vertex, where $t = (|S| \cdot (p+k+1)+1)g(a)$. Using this consecutive sequence of t sets, we will now define an x-sequence $\sigma^\star = (u_1, \dots, u_t)$ of length t. The vertices u_j in this sequence are defined inductively as follows. Let u_1 be any vertex of L_1. Then for any $j \in [t-1]$, u_{j+1} is the witness for the non dangerous vertex u_j. Note that $\sigma^\star$ might not be an x-good sequence. In what follows, we prove some nice properties of $\sigma^\star$ and then use them to refine $\sigma^\star$ to obtain an x-good sequence. We would like to remark that this refinement procedure is required to prove Property 3 of Definition 4.1. We begin by proving a claim which will lead to the refinement. We will first show that these sets are disjoint.

▶ **Lemma 4.8.** *For every $j \in [t-1]$, (u_j, u_{j+1}) is a $(V_j^{\sigma^\star}, X)$-cut in $G[C]$.*

Proof. For the sake of contradiction, suppose that this is not the case and let P be a minimal $(V_j^{\sigma^\star}, u)$-path for some $u \in X$, in $G[C] - \{u_j, u_{j+1}\}$. If $u \in L_s$ for $s \leq j-1$, then this contradicts the fact that u_{j+1} is in L_{j+1}, and if $u \in L_s$ for $s \geq j+2$, then this contradicts the fact that u is not in L_{j+1}. If $u \in L_j$, then this contradicts the fact that u_j is the witness of u_{j-1} and if $u \in L_{j+1}$, this contradicts the fact that u_{j+1} is the witness of u_j. ◀

As a corollary of Lemma 4.8, we conclude the following.

▶ **Lemma 4.9.** *For each $i, j \in [t]$, where $i \neq j$, we have $V_i^{\sigma^\star} \cap V_j^{\sigma^\star} = \emptyset$.*

Let $S_1 \subseteq S$ be the set of all those vertices s of S such that there exists $I \subseteq [t]$ of size at least $k+p+2$ and for each $b \in I$, s is adjacent to some vertex of $V_b^{\sigma^\star} \cup u_b$ (or to u_b when $b = t$). Let $S_2 = (S \setminus S_1) \setminus x$. An index $b \in [t]$ is called *affected* if there exists $s \in S_2$ such that s is adjacent to $V_b^{\sigma^\star} \cup u_b$ (or to u_b when $b = t$). By the definition of S_2, the number of affected indices is at most $|S_2| \cdot (k+p+1) \leq |S| \cdot (k+p+1)$. Since the length of $\sigma^\star$ is $t = (|S| \cdot (p+k+1)+1)g(a)$ and the number of affected indices is at most $|S|(p+k+1)$, there exists an interval σ of $\sigma^\star$ of length $g(a)$ such that none of the indices corresponding to the subscripts of the vertices in σ are affected. Without loss of generality, let $\sigma = (u_1, \ldots, u_{g(a)})$. We will now show that σ has Property 3 of Definition 4.1.

▶ **Lemma 4.10** (Property 3)**.** *Let T be some $\mathcal{F}_p$-DELETION set of size at most $k+1$ such that $x \notin T$. Then $N(C^\sigma) \cap (S \setminus T) = \{x\}$.*

Proof. By the definition of C^σ, $N(C^\sigma) \cap (S \setminus T)$ contains x. For the sake of contradiction, say $s \in S \setminus x$ belongs to $N(C^\sigma) \cap (S \setminus T)$. Since none of the indices corresponding to the vertices in σ are affected, we conclude that $s \in S_1 \setminus T$. Since $|T| \leq k+1$, σ is an interval of $\sigma^\star$ and from Lemma 4.9, there exists at least p indices in $[t]$ such that for each of these p indices, say b, $T \cap (V_b^{\sigma^\star} \cup u_b) = \emptyset$ (or $u_b \notin T$, if $b = t$) and s is a neighbour of each of these p sets $V_b^{\sigma^\star} \cup u_b$. Then the graph induced by $G - T$ on x, s and p of these sets contains $K_{2,p}$ as a minor in $G - T$, which is a contradiction as T is an $\mathcal{F}_p$-DELETION set. ◀

▶ **Lemma 4.11** (Property 2)**.** *For every $j \in [2, t-1]$, $\{u_{j-1}, u_{j+1}\}$ is a (u_j, X)-cut in C.*

Proof. Suppose there is a path in C between u_j and some vertex $u \in X$ different from $\{u_{j-1}, u_{j+1}\}$. By definition of the L_i's, u belongs to either L_{j-1}, L_j or L_{j+1}. If u belongs to L_{j-1} or L_j, this contradicts the fact that u_j is the witness of u_{j-1}. If u belongs to L_{j+1}, this contradicts the fact that u is the witness of a vertex different from u_j in L_j, so we reach a contradiction. ◀

▶ **Lemma 4.12** (Property 4)**.** $(N(C^\sigma) \setminus \{u_1, u_{g(a)}\}) \cap C \subseteq \{u_1, u_{g(a)}\}$.

Proof. Fix $i \in [g(a)-1]$. We will first show that no vertex of V_i^σ has a neighbour in $C \setminus C^\sigma$. For the sake of contradiction say $v \in V_i^\sigma$ is a neighbour of $w \in C \setminus C^\sigma$. Since w is on some walk between two vertices of $X \setminus \{u_1, \ldots, u_{g(a)}\}$, this implies that there is a path from v to a vertex in X that does not intersect $\{u_1, \ldots, u_{g(a)}\}$. This contradicts Lemma 4.8.

It remains to show that none of the vertices in $\{u_2, \ldots, u_{g(a)-1}\}$ have a neighbour in $C \setminus C^\sigma$. We show this in two parts. Note from the construction of the sets L_i, that for any $u_i \in L_i$, its neighbours in X are either in L_{i-1}, L_i or L_{i+1}. Because no vertex of L_i is dangerous and $|L_{i-1}| = |L_i|$, L_i is an independent set and the only potential neighbour of u_i in L_{i-1} and L_{i+1} is u_{i-1} and u_{i+1} respectively. Thus we conclude that no vertex in $\{u_2, \ldots, u_{g(a)-1}\}$ has a neighbour in $(C \setminus C^\sigma) \cap X$.

To finish the proof we need to show that no vertex of $\{u_2, \ldots, u_{g(a)-1}\}$ has a neighbour in $(C \setminus C^\sigma) \setminus X$. For the sake of contradiction, say u_i, for $i \in [2, g(a)-1]$, has a neighbour $w \in (C \setminus C^\sigma) \setminus X$. Since w is on some walk between two vertices of $X \setminus \{u_1, \ldots, u_{g(a)}\}$, this would imply a walk from a vertex of $X \setminus \{u_2, \ldots, u_{g(a)-1}\}$ to u_i, $i \in [2, g(a)-1]$. This either contradicts that $u_i \in L_i$ or that $w \notin C^\sigma$. ◀

From the construction of the sequence $(L_0, \ldots, L_l)$, observe that σ satisfies Property 1 of Definition 4.1. This together with Lemmas 4.10, 4.11 and 4.12 and the fact that the length of σ is $g(a)$, proves Lemma 4.3.

4.2.2 Finding an irrelevant edge

The goal of this section is to complete the proof of Lemma 3.2 using Lemma 4.3. Let $\sigma = (u_1, \ldots, u_r)$ be an x-good sequence. The *graph induced by* σ, denoted by $G[\sigma]$, is a 2-boundaried graph $G[C^\sigma]$ with boundary u_1, u_r. By $G[\sigma_i]$ we denote the 2-boundaried graph induced by the i-block of σ, with boundary u_i, u_{i+1}. Let $\hat{G} = G[\sigma] \cup x$ be a boundaried graph with boundary that is a subset of $\{x, u_1, u_r\}$. Let $h' = 2(h+3)$. The folio-set of $G[\sigma]$, denoted by folio-set$(G[\sigma])$, is the collection of $\{\cup_{i \in [r]} \{h'\text{-folio}(G[\sigma_i])\}\}$, where r is the length of σ.

In order to prove Lemma 3.2, it is enough to show that there is a vertex $u_{red} \in X$, such that for any $T \subseteq V(G)$ of size at most k, if there exists a graph H on h vertices that is a minor of $G - T$ and whose minor model uses the edge xu_{red}, then there exists a minor model of H in $G - T$ that does not use the edge xu. From Lemma 4.2 and Lemma 23 of [12] (stated below) this will follow from Lemma 4.14.

▶ **Proposition 4.13** (Lemma 23, [12]). *Let G_1 and G_2 be t-boundaried graphs and $G = G_1 \oplus G_2$. A graph H is a minor of G if and only if there exist $H_1 \leq_m G_1$ and $H_2 \leq_m G_2$ such that $|V(H_1)| \leq |V(H)| + t$, $|V(H_2)| \leq |V(H)| + t$ and $H \leq_m H_1 \oplus H_2$.*

▶ **Lemma 4.14.** *If the length of σ is $g(a)$, then one can find a vertex $u_{red} \in \sigma$ in polynomial time such that the following holds. Let H be a 3-boundaried graph with boundary $\{x\} \subseteq B \subseteq \{x, u_1, u_r\}$ of size at most $h+3$ that is present as a (boundaried) minor in $\hat{G} - T$, where $T \subseteq V(\hat{G}) \setminus x$ and $|T| \leq 3$. Then there exists a minor model of H in $\hat{G} - T$ that does not use the edge xu_{red}.*

▶ **Proposition 4.15** ([12]). *If ϕ is a minimal minor model of H in G, then every vertex in the minor model has degree at most $|V(H)|$ in the minor model.*

Let H be a 3-boundaried graph with boundary $\{x\} \subseteq B \subseteq \{x, u_1, u_r\}$ in G. Let ϕ be a minimal minor model of H in G. Let $\phi' = \phi \setminus x$ and H' be the (boundaried) graph witnessed by ϕ'. By Proposition 4.15, $|V(H')| \leq 2|V(H)|$. Let ϕ_1 and ϕ_2 be two minor models of some 2-boundaried graph H in $G[\sigma]$. Then ϕ_2 is said to be *σ-compatible* with ϕ_1, if for every branch set of ϕ_1 that has a vertex of σ, the corresponding branch set of ϕ_2 also has a vertex of σ. Recall that all the vertices of σ are the vertices of X and hence they are neighbours of x. With the discussion above, it is not difficult to see that to prove Lemma 4.14, it is enough to prove Lemma 4.16.

▶ **Lemma 4.16.** *If the length of σ is $g(a)$, then one can find a vertex $u_{red} \in \sigma$ in polynomial time such that the following holds. Let H be a 2-boundaried graph with boundary $B \subseteq \{u_1, u_r\}$ of size at most h' that is present as a (boundaried) minor in $\hat{G} - T$, where $T \subseteq V(G[\sigma])$ and $|T| \leq 3$. Let ϕ be a minor model of H in $\hat{G} - T$. Then there exists a minor model ϕ' of H in $\hat{G} - T$ that does not use the edge xu_{red} and is σ-compatible with ϕ.*

The following lemma will be crucially used in the arguments that follow. It depicts the structure of minors in $G[\sigma]$. Let H be a 2-boundaried minor in $G[\sigma]$. Let ϕ be a minor model of H in $G[\sigma]$. The *blocks of σ used by ϕ* refers to the collection of blocks of σ that have a non-empty intersection with the vertices of the minor model ϕ. The *crucial blocks used by ϕ* refers to those blocks which have a branch set of ϕ fully contained in it. Note that the number of crucial blocks of ϕ is at most $|V(H)|$.

▶ **Lemma 4.17** ($\star$). *Let H be a connected 2-boundaried minor in $G[\sigma]$. Let ϕ be some minor model of H in $G[\sigma]$. Then there exists a minor model ϕ' of H in $G[\sigma]$ obtained from ϕ by replacing the vertices in the non-crucial blocks used by ϕ with any arbitrary path between the endpoints of the block.*

▶ **Definition 4.18** (Chunk partition of σ). *Let σ be an x-good sequence. A* chunk *of σ is an interval of σ. A* chunk partition *of σ is a partition of the blocks of σ into intervals. Let σ' be a chunk of σ. Then folio-set(σ') is a set containing the h'- folio($G[\sigma_i]$), for each i-block in σ'. A chunk partition of σ is called* uniform *if the folio-sets of all chunks in the partition are same. In particular, for any $j \in [a]$, a chunk partition of σ is called j-*uniform *if it is uniform and the size of the folio-set of each chunk is exactly j.*

▶ **Lemma 4.19** ($\star$, Finding an i-uniform chunk partition). *Let σ be an x-good sequence and let the size of the folio-set(σ) be exactly i. If the length of σ is $g(i)$ then there exists an interval of σ, say σ', which admits a j-uniform chunk partition of length $18h'$, for some $j \in [i]$.*

▶ **Lemma 4.20** ($\star$, Replacement Lemma). *Let σ be an x-good sequence and $\chi = (\chi_1, \dots, \chi_s)$ be an i-uniform chunk partition of σ, for some $i \in [a]$. Let $s \geq h'$. Let H be a 2-boundaried minor in $G[\sigma]$ of size at most h'. Then H is present as a 2-boundaried minor in every graph that is induced on any h' sized interval of χ. Moreover, the later minor model of H is σ-compatible with the former minor model of H.*

▶ **Lemma 4.21** ($\star$). *Let σ be an x-good sequence of length $g(a)$ that admits an i-uniform chunk partition, for some $i \in [a]$, of length $18h'$. Then Lemma 4.16 holds.*

From Lemmas 4.21 and 4.19, Lemma 4.16 follows. Lemma 4.16 together with Lemma 4.3 finishes the proof of Lemma 3.2.

5 Conclusion

In this article we showed that $\mathcal{F}$-Deletion where all graphs in $\mathcal{F}$ are connected and $\mathcal{F}$ contains $K_{2,p}$ admits a uniform polynomial kernel of size $\mathcal{O}(k^{10})$. This result is the third example where $\mathcal{F}$-Deletion admits a uniform polynomial kernel; the first two being the Treedepth-η Deletion and the case when $\mathcal{F}$ contains θ_p. The most interesting aspect of our result is defining and obtaining an extremely structured set of vertices that have a small effective boundary. This structure is exploited to reduce the degree of the vertices to $k^{\mathcal{O}(1)}$.

We conclude with some intriguing open questions. Our result does not extend to the case when $\mathcal{F}$ is allowed to contain disconnected graphs. The first question is: can one obtain a uniform polynomial kernel when $\mathcal{F}$ contains $K_{2,p}$ and other possibly disconnected graphs? In fact, handling disconnected graphs in the kernelization algorithm of [12] is one point which introduces non-uniform bounds. Can this step of the kernelization algorithm of [12] be made to work without introducing non-uniformity? Or even more specifically, can we find some non-trivial families $\mathcal{F}$ which contain disconnected graphs but admit uniform polynomial kernels? Lastly, can we characterize the families $\mathcal{F}$ that admit a uniform polynomial kernel?

As long as we do not resolve the last question completely, one would be interested in finding more and more non-trivial families for which the problem admits a uniform polynomial kernel.

References

1 Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 641–650. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347153.

2 Hans L Bodlaender, Fedor V Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M Thilikos. (Meta) kernelization. *Journal of the ACM (JACM)*, 63(5):1–69, 2016. doi:10.1145/2973749.

3 Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM (JACM)*, 63(5):1–65, 2016. doi:10.1145/2820609.

4 Guantao Chen, Yoshimi Egawa, Ken-ichi Kawarabayashi, Bojan Mohar, and Katsuhiro Ota. Toughness of-minor-free graphs. *The Electronic Journal of Combinatorics [electronic only]*, 18(1):Research–Paper, 2011.

5 Guantao Chen, Laura Sheppardson, Xingxing Yu, and Wenan Zang. The circumference of a graph with no $K_{3,t}$-minor. *Journal of Combinatorial Theory, Series B*, 96(6):822–845, 2006. doi:10.1016/J.JCTB.2006.02.006.

6 Maria Chudnovsky, Bruce Reed, and Paul Seymour. The edge-density for $K_{2,t}$ minors. *Journal of Combinatorial Theory, Series B*, 101(1):18–46, 2011. doi:10.1016/J.JCTB.2010.09.001.

7 Youssou Dieng. *Décomposition arborescente des graphes planaires et routage compact.* PhD thesis, Bordeaux 1, 2009.

8 Reinhard Diestel. *Graph theory.* Springer Publishing Company, Incorporated, 2018.

9 Guoli Ding. Graphs without large $K_{2,n}$-minors. *arXiv preprint*, 2017. arXiv:1702.01355.

10 Mark N Ellingham, Emily A Marshall, Kenta Ozeki, and Shoichi Tsuchiya. A characterization of $K_{2,4}$-minor-free graphs. *SIAM Journal on Discrete Mathematics*, 30(2):955–975, 2016. doi:10.1137/140986517.

11 Fedor V Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM Journal on Discrete Mathematics*, 30(1):383–410, 2016. doi:10.1137/140997889.

12 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. See http://www.ii.uib.no/~daniello/papers/PFDFullV1.pdf for the full version. doi:10.1109/FOCS.2012.62.

13 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing.* Cambridge University Press, 2019.

14 Archontia C Giannopoulou, Bart MP Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Transactions on Algorithms (TALG)*, 13(3):1–35, 2017. doi:10.1145/3029051.

15 Bart M. P. Jansen and Michal Wlodarczyk. Lossy planarization: a constant-factor approximate kernelization for planar vertex deletion. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 900–913. ACM, 2022. doi:10.1145/3519935.3520021.

16 Bart MP Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1802–1811. SIAM, 2014.

17 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 639–648. IEEE, 2009. doi:10.1109/FOCS.2009.45.

18 John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

19 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/S00453-010-9484-Z.

20 Joseph Samuel Myers. The extremal function for unbalanced bipartite minors. *Discrete mathematics*, 271(1-3):209–222, 2003. doi:10.1016/S0012-365X(03)00051-7.

21 Katsuhiro Ota and Kenta Ozeki. Spanning trees in 3-connected $K_{3,t}$-minor-free graphs. *Journal of Combinatorial Theory, Series B*, 102(5):1179–1188, 2012. doi:10.1016/J.JCTB.2012.07.002.

22 Neil Robertson and PD Seymour. Graph minors. Xlll. The disjoint paths problem. *J. Combin. Theory Ser. B*, 63:65–110, 1995.

23 Noah Streib and Stephen J Young. Dimension and structure for a poset of graph minors. *University of Louisville Department of Mathematics*, 2010.

Complexity Framework for Forbidden Subgraphs II: Edge Subdivision and the “H”-Graphs

Vadim Lozin
University of Warwick, Coventry, UK

Barnaby Martin
Durham University, UK

Sukanya Pandey
Utrecht University, The Netherlands

Daniël Paulusma
Durham University, UK

Mark Siggers
Kyungpook National University, Daegu, Republic of Korea

Siani Smith
University of Bristol, UK
Heilbronn Institute for Mathematical Research, Germany

Erik Jan van Leeuwen
Utrecht University, The Netherlands

Abstract

For a fixed set $\mathcal{H}$ of graphs, a graph G is $\mathcal{H}$-subgraph-free if G does not contain any $H \in \mathcal{H}$ as a (not necessarily induced) subgraph. A recent framework gives a complete classification on $\mathcal{H}$-subgraph-free graphs (for finite sets $\mathcal{H}$) for problems that are solvable in polynomial time on graph classes of bounded treewidth, NP-complete on subcubic graphs, and whose NP-hardness is preserved under edge subdivision. While a lot of problems satisfy these conditions, there are also many problems that do not satisfy all three conditions and for which the complexity in $\mathcal{H}$-subgraph-free graphs is unknown. We study problems for which only the first two conditions of the framework hold (they are solvable in polynomial time on classes of bounded treewidth and NP-complete on subcubic graphs, but NP-hardness is not preserved under edge subdivision). In particular, we make inroads into the classification of the complexity of four such problems: Hamilton Cycle, k-Induced Disjoint Paths, C_5-Colouring and Star 3-Colouring. Although we do not complete the classifications, we show that the boundary between polynomial time and NP-complete differs among our problems and also from problems that do satisfy all three conditions of the framework, in particular when we forbid certain subdivisions of the “H”-graph (the graph that looks like the letter “H”). Hence, we exhibit a rich complexity landscape among problems for $\mathcal{H}$-subgraph-free graph classes.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases forbidden subgraph, complexity dichotomy, edge subdivision, treewidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.47

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2211.14214`

Acknowledgements We are grateful to Matthew Johnson, Jelle J. Oostveen and Hans Bodlaender for useful discussions.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 47; pp. 47:1–47:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph containment relations, such as the (topological) minor and induced subgraph relations, have been extensively studied both from a graph-structural and algorithmic point of view. In this paper, we focus on the *subgraph relation.* If a graph H can be obtained from a graph G by a sequence of vertex deletions and edge deletions, then G contains H as a *subgraph*; otherwise, G is *H-subgraph-free.* For a set of graphs $\mathcal{H}$, a graph G is *$\mathcal{H}$-subgraph-free* if G is H-subgraph-free for every $H \in \mathcal{H}$; if $\mathcal{H} = \{H_1, \ldots, H_p\}$, then we also write that G is $(H_1, \ldots, H_p)$-subgraph-free. Graph classes closed under deletion of edge and vertices are called *monotone* [2, 8], and every monotone graph class $\mathcal{G}$ can be characterized by a unique (and possibly infinite) set of forbidden induced subgraphs $\mathcal{H}_{\mathcal{G}}$. We determine the complexity of two connectivity problems Hamilton Cycle and k-Induced Disjoint Paths, and two colouring problems C_5-Colouring and Star 3-Colouring on $\mathcal{H}$-subgraph-free graphs for various families $\mathcal{H}$. We focus on families $\mathcal{H}$ consisting of certain *subdivided "H"-graphs* $\mathbb{H}_i$, where $\mathbb{H}_1$ looks like the letter "H" (see Fig. 1 for the definition of the graphs $\mathbb{H}_i$). At first sight, these problems appear to have not much in common. Moreover, the graphs $\mathbb{H}_i$ might also seem arbitrary. However, these problems turn out to be well suited for a combined study, as they fit in a more general framework, in which the graphs $\mathbb{H}_i$ play a crucial role.

Context

If a graph problem is computationally hard, it is natural to restrict the input to some special graph class. Ideally we would like to know exactly which properties P such a graph class $\mathcal{G}$ must have such that any hard graph problem that satisfies some conditions C becomes easy on $\mathcal{G}$. The distinction between "*easy*" and "*hard*" means, in this paper, P versus NP-complete, but could also mean P versus Π^P_{2k}-complete [13], or almost-linear versus at-least-quadratic [20]. We first discuss some natural conditions C.

A graph is *subcubic* if every vertex has degree at most 3, or equivalently if is $K_{1,4}$-subgraph-free, where $K_{1,4}$ denotes the 5-vertex star. For $p \geq 1$, the *p-subdivision* of an edge $e = uv$ of a graph G replaces e by a path of $p+1$ edges with endpoints u and v. The *p-subdivision* of a graph G is the graph obtained from G after p-subdividing each edge; see also Fig. 1. For a graph class $\mathcal{G}$ and an integer p, we let $\mathcal{G}^p$ be the class consisting of the p-subdivisions of the graphs in $\mathcal{G}$. A graph problem Π is hard *under edge subdivision of subcubic graphs* if for every $j \geq 1$ there is an $\ell \geq j$ such that: if Π is hard for the class $\mathcal{G}$ of subcubic graphs, then Π is hard for $\mathcal{G}^\ell$. We can now say that a graph problem Π has property:

- C1 if Π is easy for every graph class of bounded tree-width;
- C2 if Π is hard for subcubic graphs (or equivalently, $K_{1,4}$-subgraph-free graphs);
- C3 if Π is hard under edge subdivision of subcubic graphs;
- C4 if Π is hard for planar graphs;
- C5 if Π is hard for planar subcubic graphs.

We say that Π is a C123-problem if it satisfies C1, C2 and C3, while for example Π is a C1$\cancel{3}$-problem if it satisfies C1 but not C3, and so on.

Classical results of Robertson and Seymour [29] yield the following two *meta-classifications.* For all sets $\mathcal{H}$, a C14-problem Π is easy on $\mathcal{H}$-minor-free graphs if $\mathcal{H}$ contains a planar graph, or else it is hard. For all sets $\mathcal{H}$, a C15-problem Π is easy on $\mathcal{H}$-topological-minor-free graphs if $\mathcal{H}$ contains a planar subcubic graph, or else it is hard. No meta-classification for the induced subgraph relation exists (apart from a limited one [20] that is a direct consequence of the treewidth dichotomy [26]). However, for the subgraph relation, known results on Independent Set [3], Dominating Set [3], Long Path [3], Max-Cut [22] and List

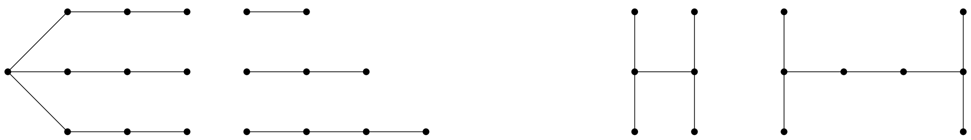


Figure 1 [7] Left: A graph in $\mathcal{S}$: the graph $S_{3,3,3} + P_2 + P_3 + P_4$, where $S_{3,3,3}$ is the 2-subdivision of the claw $K_{1,3}$. Right: the "H"-graph $\mathbb{H}_1$ and the graph $\mathbb{H}_3$; for $i \geq 2$, the graph $\mathbb{H}_i$ ($i \geq 2$) is obtained from $\mathbb{H}_1$ by $(i-1)$-subdividing the edge that joins the middle vertices of the two P_3s.

Colouring [17] for monotone graph classes that are *finitely defined* (so, where the associated set of forbidden subgraphs $\mathcal{H}$ is finite) were recently unified and extended in [20]. This led to a new meta-classification, where the set $\mathcal{S}$ consists of all graphs, in which every connected component is either a path or a subcubic tree with exactly one vertex of degree 3 (see Fig. 1).

▶ **Theorem 1** ([20]). *For any* finite *set of graphs $\mathcal{H}$, a* C123*-problem* Π *is easy on $\mathcal{H}$-subgraph-free graphs if $\mathcal{H}$ contains a graph from $\mathcal{S}$, or else it is hard.*

The easy part of Theorem 1 holds because a class of $\mathcal{H}$-subgraph-free graphs satisfies C1 if and only if $\mathcal{H}$ contains a graph from $\mathcal{S}$ [28]. The hard part follows from combining C2 and C3, as discussed below. In [20], 20 C123-problems were identified on top of the five above.

Our Focus

Many graph problems are not C123. See [6] and [7] for partial complexity classifications of the C$\cancel{1}$23-problems Subgraph Isomorphism and Steiner Forest, respectively, for H-subgraph-free graphs and [21] for partial complexity classifications of the C1$\cancel{2}$3-problems (Independent) Feedback Vertex Set, Connected Vertex Cover, Colouring (see also [18]) and Matching Cut for H-subgraph-free graphs (note that if a problem does not satisfy C2, then C3 is implied). Here, we consider the question: *Can we classify the complexity of C12$\cancel{3}$-problems on monotone graph classes?*

Why the Graphs $\mathbb{H}_i$

All C1-problems are easy on $\mathcal{H}$-subgraph-free graphs if $\mathcal{H}$ has a graph from $\mathcal{S}$ [28]. The infinite set $\mathcal{M} = \{C_3, C_4, \ldots, K_{1,4}, \mathbb{H}_1, \mathbb{H}_2, \ldots\}$ of minimal graphs not in $\mathcal{S}$ is a maximal antichain in the poset of connected graphs under the subgraph relation. Conditions C2 and C3 ensure that for every finite set $\mathcal{M}'$, C123-problems are hard on $\mathcal{M}'$-subgraph-free graphs if $\mathcal{M}' \subseteq \mathcal{M}$. If C3 is not satisfied, this is no longer guaranteed. Hence, a natural starting point to answer our research question is to determine for which finite subsets $\mathcal{M}' \subseteq \mathcal{M}$, C12-problems are still easy on $\mathcal{M}'$-subgraph-free graphs. So consider a C12-problem Π that is not C3. Let $\mathcal{M}'$ be a finite subset of $\mathcal{M}$. If $\mathcal{M}' = \{K_{1,4}\}$, then Π is hard for $\mathcal{M}'$-subgraph-free graphs due to C2. Hence, $\mathcal{M}'$ must contain at least one C_s or $\mathbb{H}_i$. The *girth* of a graph (that is not a forest) is the length of a shortest cycle in it. We say that Π has property:

- C2' if for all $g \geq 3$, Π is hard for subcubic graphs of girth at least g.

A graph is subcubic and of girth $g \geq 4$ if and only if it is $(K_{1,4}, C_3, \ldots, C_{g-1})$-subgraph-free. So if Π is not only C12, but even a C12'-problem, then Π is hard on $\mathcal{M}'$-subgraph-free graphs unless $\mathcal{M}'$ contains some $\mathbb{H}_i$. This makes studying the graphs $\mathbb{H}_i$ even more pressing.

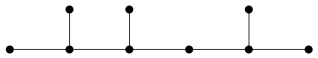


Figure 2 The tree T.

Our Testbed Problems

We take, as mentioned, the following four testbed problems:

(i) Hamilton Cycle, which is to decide if a graph G has a *Hamiltonian cycle*, i.e., a cycle through all vertices of G. This problem satisfies C1 [4], and it is NP-complete even for bipartite subcubic graphs of girth g, for every $g \geq 3$ [2]. Hence, it is even a C12'-problem.

(ii) k-Induced Disjoint Paths, which is to decide, given a graph G and pairwise disjoint vertex pairs $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k)$ for some fixed $k \geq 2$, if G has k *mutually induced* s_i-t_i-paths P^i, i.e., $P^1, \ldots, P^k$ are pairwise vertex-disjoint and there are no edges between vertices from different P^i and P^j. For every $k \geq 2$, this problem satisfies C1 due to Courcelle's Theorem [10] and also satisfies C2 [24]. Hence, it is a C12-problem for all $k \geq 2$.

(iii) C_5-Colouring, which is to decide if a graph G has a *homomorphism* (C_5-*colouring*) to the 5-cycle C_5 , i.e., a mapping $f : V(G) \to V(C_5)$ such that for every $uv \in E(G)$, it holds that $f(u)f(v) \in E(C_5)$. The problem satisfies C1 [12] and C2 [15]. Hence, it is a C12-problem.

(iv) Star 3-Colouring, which is to decide if a graph G has a *star* 3-*colouring*, i.e., a mapping $f : V(G) \to \{1, 2, 3\}$ such that for every i, the set U_i of vertices of G mapped to i is independent (so, f is a 3-*colouring*), and moreover, $U_1 \cup U_2$, $U_1 \cup U_3$, $U_2 \cup U_3$ all induce a disjoint union of stars. The problem satisfies C1 due to Courcelle's Theorem [10], and it is NP-complete for bipartite planar subcubic graphs of girth at least g, for every $g \geq 3$ [31]. Hence, it is even a C12'-problem.

We do not know if k-Induced Disjoint Paths and C_5-Colouring are C12', even though C_5-Colouring is NP-complete for graphs of maximum degree $6 \cdot 5^{13}$ and girth at least g, for all $g \geq 3$ (see Section 2.1).

All four problems violate C3. For $p \geq 3$, C_5-Colouring and Star 3-Colouring become true (all yes-instances) under p-subdivision, while Hamilton Cycle becomes false (all no-instances, unless we started with a cycle), and k-Induced Disjoint Paths reduces to the polynomial-time solvable problem k-Disjoint Paths [30, 33], which only requires the paths in a solution to be pairwise vertex-disjoint. See Section 3. We also note the following. First, when k is part of the input, Disjoint Paths and Induced Disjoint Paths are C123-problems [20]. Second, instead of C_5-Colouring we could have considered C_{2i+1}-Colouring, which is a C12-problem for all $i \geq 2$ [12, 15]. Third, Star-k-Colouring does not satisfy C2 for large k, as all subcubic graphs are star 10-colourable (as shown in Section 3).

Our Results

We show that the complexity of our four problems differ from each other and also from C123-problems, when we forbid certain graphs $\mathbb{H}_i$. We first show that C1-problems, and thus C12-problems, are easy on $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$-subgraph-free graphs for every $\ell \geq 1$ and on $(\mathbb{H}_i, \mathbb{H}_{2i}, \mathbb{H}_{3i}, \ldots)$-subgraph-free graphs for every $i \geq 1$ (so, in particular if we forbid all even $\mathbb{H}_i$), as all these graph classes have bounded treewidth, as we show in Section 4. In contrast, any hard problem for bipartite graphs in which one partition class has maximum degree 2 is hard on $(\mathbb{H}_1, \mathbb{H}_3, \ldots)$-subgraph-free graphs (so, if we forbid all odd $\mathbb{H}_i$): every path between vertices of degree at least 3 has even length. The NP-hardness reduction in [1] shows that Star 3-Colouring is such a problem (see also Section 2.2).

The above results immediately give us Theorem 5. For the other three problems, we prove additional results. In Section 5 we show that Hamilton Cycle is polynomial-time solvable for $\mathbb{H}_\ell$-subgraph-free graphs if $\ell = 3$ by doing this for the superclass of T-subgraph-free graphs (T is the tree shown in Figure 2). For $\ell \in \{1, 2\}$ this was proven in [25]. On a side note, there exist trees T^* for which Hamilton Cycle is NP-complete over T^*-subgraph-free graphs. We refer to [23, 25] for examples of such trees T^*, which are not subdivided “H”-graphs $\mathbb{H}_i$.

In Section 6 we prove that for all $k \geq 2$, k-Induced Disjoint Paths is polynomial-time solvable for $\mathbb{H}_\ell$-subgraph-free graphs for $\ell \in \{1, 2\}$, but NP-complete for subcubic $(\mathbb{H}_4, \ldots, \mathbb{H}_\ell)$-subgraph-free graphs for all $\ell \geq 4$. For the first result, we first apply the algorithm for k-Disjoint Paths [30]. If this yields a solution that is not mutually induced, we apply a reduction rule and repeat the process on a smaller instance. For the second result, we carefully adapt the proof of [24] that shows that the problem of deciding if a subcubic graph contains an induced cycle between two given degree 2-vertices is NP-complete.

In Section 7 we determine all C_5-*critical* $\mathbb{H}_3$-subgraph-free graphs, which are not C_5-colourable unlike every proper subgraph of them. We show that this leads to a polynomial-time algorithm for $\mathbb{H}_3$-subgraph-free graphs that is even *certifying*. In contrast, the problem is NP-complete for the “complementary” class of $(\mathbb{H}_1, \mathbb{H}_2, \mathbb{H}_4, \mathbb{H}_5, \mathbb{H}_7, \mathbb{H}_8, \ldots)$-subgraph-free graphs (see Section 2.3).

The above results yields the following state-of-the-art summaries:

▶ **Theorem 2.** Hamilton Cycle *is polynomial-time solvable for* $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$*-subgraph-free graphs* $(\ell \geq 1)$*, for* $(\mathbb{H}_i, \mathbb{H}_{2i}, \mathbb{H}_{3i}, \ldots)$*-subgraph-free graphs* $(i \geq 1)$ *and for* $\mathbb{H}_\ell$*-subgraph-free graphs* $(\ell \in \{1, 2, 3\})$*.*

▶ **Theorem 3.** *For all* $k \geq 2$*,* k-Induced Disjoint Paths *is polynomial-time solvable for* $\mathbb{H}_\ell$*-subgraph-free graphs* $(\ell \in \{1, 2\})$*, for* $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$*-subgraph-free graphs* $(\ell \geq 1)$ *and for* $(\mathbb{H}_i, \mathbb{H}_{2i}, \mathbb{H}_{3i}, \ldots)$*-subgraph-free graphs* $(i \geq 1)$*, but* NP*-complete for subcubic* $(\mathbb{H}_4, \ldots, \mathbb{H}_\ell)$*-subgraph-free graphs* $(\ell \geq 4)$*.*

▶ **Theorem 4.** C_5-Colouring *is polynomial-time solvable for* $\mathbb{H}_3$*-subgraph-free graphs, for* $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$*-subgraph-free graphs* $(\ell \geq 1)$ *and for* $(\mathbb{H}_i, \mathbb{H}_{2i}, \mathbb{H}_{3i}, \ldots)$*-subgraph-free graphs* $(i \geq 1)$*, but* NP*-complete for* $(\mathbb{H}_1, \mathbb{H}_2, \mathbb{H}_4, \mathbb{H}_5, \mathbb{H}_7, \mathbb{H}_8, \ldots)$*-subgraph-free graphs.*

▶ **Theorem 5.** Star 3-Colouring *is polynomial-time solvable for* $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$*-subgraph-free graphs* $(\ell \geq 1)$ *and* $(\mathbb{H}_i, \mathbb{H}_{2i}, \mathbb{H}_{3i}, \ldots)$*-subgraph-free graphs* $(i \geq 1)$*, but* NP*-complete for* $(\mathbb{H}_1, \mathbb{H}_3, \mathbb{H}_5 \ldots)$*-subgraph-free graphs.*

We note that the complexity classifications above indeed differ except perhaps for Hamilton Cycle and k-Induced Disjoint Paths. Hence, Theorems 2–5 give clear evidence of a rich landscape for C12-problems on $\mathcal{H}$-subgraph-free graphs. In Section 8 we discuss open problems resulting from our study.

2 Some Basic Results

In this section, we provide further details for some statements made in Section 1.

2.1 C_5-Colouring for Bounded Degree and Large Girth

The k-Colouring problem is to decide if a graph G has a *k-colouring*, which is a mapping $c : V(G) \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ for any two adjacent vertices u and v of G. We need a result of Emden-Weinert, Hougardy and Kreuter:

▶ **Theorem 6** ([14]). *For all $k \geq 3$ and all $g \geq 3$, k-Colouring is NP-complete for graphs with girth at least g and with maximum degree at most $6k^{13}$.*

We now repeat the proof of Chudnovsky et al. [9], which comes down to replacing each edge of an input graph G of 5-Colouring, which we may assume has girth at least g and maximum degree at most $6 \cdot 5^{13}$ due to Theorem 6, by a path of length 3. This yields a new graph G' of girth at least g, such that G and G' have the same maximum degree. Hence, we derive the following result.

▶ **Proposition 7.** *For every $g \geq 3$, C_5-Colouring is NP-complete for graphs with girth at least g and with maximum degree at most $6 \cdot 5^{13}$.*

2.2 The Standard NP-hardness Reduction to Star-3-Colouring

For reference, we explain the gadget from Albertson et al. [1] that yields the following result.

▶ **Theorem 8** ([1]). *Star 3-Colouring is NP-complete for planar bipartite graphs in which one partition class has size 2.*

Proof. Reduce from 3-Colourability which is known to be NP-complete even for planar graphs [11]. Let G be a planar graph. Replace each edge e by three new vertices a_e, b_e, c_e that are made adjacent only to the two end-vertices of e in G. Let G' be the resulting graph. Then every vertex of $V(G') \setminus V(G)$ has degree 2 in G. Moreover, G' is planar and bipartite with partition classes $V(G') \setminus V(G)$ and $V(G)$. It remains to observe that G has a 3-colouring if and only if G' has a star 3-colouring. ◀

2.3 The Standard NP-hardness Reduction to C_5-Colouring

We make the following observation.

▶ **Proposition 9.** *C_5-Colouring is NP-complete for $(\mathbb{H}_1, \mathbb{H}_2, \mathbb{H}_4, \mathbb{H}_5, \ldots)$-subgraph-free graphs.*

Proof. It is well known [19] and easy to see that there is a reduction from K_5-Colouring, which is to decide if a graph has a K_5-colouring, that is, a homomorphism from G to the complete graph K_5 on five vertices. This problem is well known to be NP-complete [19]. Let G be a graph, and let G' be the 2-subdivision of G. We note that G' is $(\mathbb{H}_1, \mathbb{H}_2, \mathbb{H}_4, \mathbb{H}_5, \ldots)$-subgraph-free (but may contain many instances of $\mathbb{H}_\ell$ where $\ell = 0 \mod 3$). Moreover, G has a K_5-colouring if and only if G' has a C_5-colouring. ◀

3 The Four Testbed Problems Do Not Satisfy C3

In this section we show that none of our four problems satisfy C3. We use the following notation in this section: for a graph G and an integer $p \geq 1$, let G_p be the p-subdivision of G (which we recall is the graph obtained from G after subdividing each edge of G exactly p times).

▶ **Proposition 10.** Hamilton Cycle *does not satisfy C3.*

Proof. We observe that for every graphs G and every $p \geq 1$, G_p is a no-instance of Hamilton Cycle unless G was a cycle. ◀

▶ **Proposition 11.** k-Induced Disjoint Paths *does not satisfy C3.*

Proof. Under any kind of subdivision, k-Induced Disjoint Paths reduces to k-Disjoint Paths over the same graph, which is in P for all $k \geq 2$, as shown in [33] for $k = 2$ and in [30] for every $k \geq 3$. ◀

▶ **Proposition 12.** C_5-Colouring *does not satisfy C3.*

Proof. We first prove that for all $p \geq 4$, and for all $x, y \in V(C_5)$, there is a walk of length p in C_5 from x to y. First let $p = 4$. To walk a distance of zero: walk two forward then two back. To walk at distance one (without loss of generality) forward: walk four backward. To walk at distance two (without loss of generality) forward: walk one back, one forward, and two forward. Now let $p = 5$. To walk a distance of zero: walk five forward. To walk at distance one (without loss of generality) forward: walk two forward, two back and one forward. To walk at distance two (without loss of generality) forward: walk one back, one forward, and three back. Finally, let $p \geq 6$. Keep moving one forward then one back until one of the two previous cases applies.

Now let G be a graph. We give each vertex in G a label from $\{1, \dots, 5\}$. From the above it follows that for every $p \geq 3$, we can extend c to a homomorphism from G_p to C_5; in other words, G_p is a yes-instance of C_5-Colouring. ◀

▶ **Proposition 13.** Star 3-Colouring *does not satisfy C3.*

Proof. Let G be a graph. We show that for all $p \geq 3$, G_p is a yes-instance of Star 3-Colouring. We do this by giving each vertex in G a label from $\{1, 2, 3\}$. The resulting labelling c might not be a 3-colouring, but this is not important: we will show that we can extend c to a star 3-colouring of G_p as follows.

Consider an edge e in G and let P be the corresponding path (of $p + 1$ edges) in G_p. It suffices to give two star 3-colourings of this path, so that the first three vertices are distinct colours and the last three vertices are distinct colours: one in which the first and last vertices are the same colour and one in which they are a different colour. Let us identify a 3-colouring of P by a sequence of length $p + 1$ over $\{1, 2, 3\}$. If $p + 1$ is a multiple of three, then use $(123)^{\frac{p+1}{3}}$ for the different colour and $(123)^{\frac{p+1}{3}-1}231$ for the same colour. If $p + 1$ is 1 mod 3, then use $(123)^{\frac{p}{3}-1}2132$ for the different colour and $(123)^{\frac{p}{3}}1$ for the same colour. If $p + 1$ is 2 mod 3, then use $(123)^{\frac{p-1}{3}}12$ for the different colour and $(123)^{\frac{p-1}{3}}21$ for the same colour. ◀

We finish this section with another small observation. Namely, we cannot generalise our result for Star 3-Colouring to Star k-Colouring for any $k \geq 3$, as for large k the problem no longer satisfies C2. In fact, we prove even a stronger statement. A k-colouring of a graph G is said to be *injective* if for every vertex $u \in V(G)$, every neighbour of u is assigned a different colour, or in other words, the union of any two colour-classes induce a disjoint union of isolated vertices and edges. So, any injective k-colouring is a star k-colouring (but the reverse does not necessarily hold, for instance the P_3 is star 2-colourable but has no injective 2-colouring).

▶ **Proposition 14.** *For $k \geq 10$, all subcubic graphs have an injective 10-colouring.*

Proof. It suffices to prove the statement for $k = 10$. We do this by induction. For the base case, a graph with one vertex is star 10-colourable. Now take a vertex v in a graph G and assume $G \setminus \{v\}$ has an injective 10-colouring. As G is subcubic, v has at most three neighbours, each of which have at most two more neighbours each. Thus there are at most nine vertices whose colour we wish to avoid. As we have ten colours in total, this means that we can safely colour v. ◀

4 Bounded Treewidth Results

A graph G contains H as a *minor* if G can be modified to H by a sequence of vertex deletions, edge deletions and edge contractions; if not, then G is *H-minor-free.*

▶ **Proposition 15.** *For every $\ell \geq 1$, the class of $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$-subgraph-free graphs has bounded treewidth.*

Proof. For $\ell \geq 1$, a $(\mathbb{H}_\ell, \mathbb{H}_{\ell+1}, \ldots)$-subgraph-free graph is $\mathbb{H}_\ell$-minor-free. For every forest F, all F-minor-free graphs have pathwidth, and thus treewidth, at most $|V(F)| - 2$ [5]. ◀

▶ **Proposition 16.** *For every $n \geq 1$, the class of $(\mathbb{H}_n, \mathbb{H}_{2n}, \mathbb{H}_{3n}, \ldots)$-subgraph-free graphs has bounded treewidth.*

Proof. If a class of graphs has unbounded treewidth, then every grid appears as a minor in some graph [29]. Let us explain the argument for $n = 2$ first. We consider that the 3×3-grid appears as a minor in some graph G in our class and let f be the minor map from G to the 3×3-grid. Consider the three vertices in the 3×3-grid that form the central row as u, v, w (in succession). Choose $u' \in f^{-1}(u), v' \in f^{-1}(v), w' \in f^{-1}(w)$ so that u', v', w' have degree greater than 2, noting that such vertices must exist. If the distance in G between u' and v' is even, of length $2i$, then there is an $\mathbb{H}_{2i}$ subgraph in G with central path from u' to v'. If the distance in G between v' and w' is even, of length $2i$, then there is an $\mathbb{H}_{2i}$ subgraph in G with central path from v' to w'. Else, there is a path of even length $4i$ from u' to w' and then there is an $\mathbb{H}_{4i}$ subgraph in G with central path from u' to w'.

For $(\mathbb{H}_n, \mathbb{H}_{2n}, \ldots)$-subgraph-free graphs, we consider the Abelian group $(\mathbb{Z}/n\mathbb{Z})$. The Davenport constant of an Abelian group G is the minimum d so that any sequence of elements of G contains a non-empty consecutive subsequence of zero-sum (that adds to the identity element 0). It is known that for $(\mathbb{Z}/n\mathbb{Z})$ the Davenport constant is n (see page 24 in [16]). Take an $(n+1) \times (n+1)$-grid and consider some row not at the top or bottom of the grid with vertices $w_1, \ldots, w_{n+1}$ in succession. Consider some $w'_1 \in f^{-1}(w_1), \ldots, w'_{n+1} \in f^{-1}(w_{n+1})$ where f is the minor map as before, and the distances x_i between w'_{i+1} and w'_i. Using the Davenport constant, there is a subsequence $x_j, \ldots, x_{j'}$ $(j' > j)$ such that $x_j + \ldots + x_{j'} = 0 \bmod n$. Now choose $w'_j, \ldots, w'_{j'+1}$ as the central path in some $\mathbb{H}_{in}$. ◀

5 Hamilton Cycle

In this section we show Theorem 17. Due to the page limit we have omitted the proofs of some of the claims in the proof of Theorem 17.

▶ **Theorem 17.** Hamilton Cycle *is polynomial-time solvable for T-subgraph-free graphs.*

Proof. Let G be a T-subgraph-free graph. We call vertices of degree 2 in G *white* and vertices of degree at least 3 *black*. The *black graph* is a subgraph of G induced by black vertices and a *black component* is a connected component in the black graph.

We first describe some helpful rules to solve the problem and a set of reductions simplifying the input graph, i.e. reductions transforming G into a graph G' that has fewer edges and/or vertices and that has a Hamiltonian cycle if and only if G has. We emphasize that by deleting an edge or a vertex from an H-subgraph-free graph, we obtain an H-subgraph-free graph again.

We start with some obvious rules:

(R1) if the graph has vertices of degree 0 or 1, then stop: G has no Hamiltonian cycle.

(R2) if the graph contains a vertex adjacent to more than two white vertices, then stop: G has no Hamiltonian cycle.

(R3) if the graph is disconnected, then stop: G has no Hamiltonian cycle.

(R4) if the graph contains a vertex v adjacent to exactly two white vertices, then delete the edges connecting v to all other its neighbours (if there are any).

Now we introduce a reduction applicable to a graph G containing an induced subgraph shown on the left in Figure 3, in which vertices a, b, c have degree 3 in G. The reduction depends on the degree of x. If the degree of x is also 3, the reduction consists in deleting the edges ab and xc. Otherwise, it transforms the graph as shown in Figure 3. We refer to this reduction as the *diamond reduction* and denote it by (R5).

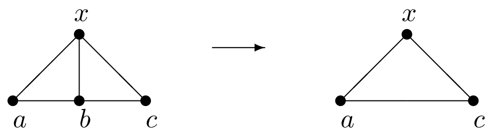


Figure 3 The diamond reduction: it is applicable to a graph G containing an induced subgraph shown on the left, in which vertices a, b, c have degree 3 in G. If the degree of x is also 3, the reduction consists in deleting the edges ab and xc. Otherwise, the reduction consists in deleting vertex b and introducing the edge ac.

We omit the proof of the next two claims.

▷ Claim 18. Let G' be a graph obtained from G by the diamond reduction. Then G has a Hamiltonian cycle if and only if G' has a Hamiltonian cycle. Moreover, if G is T-subgraph-free, then so is G'.

In Figure 4, we illustrate the *butterfly reduction* denoted by (R6).

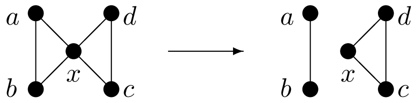


Figure 4 The butterfly reduction: it is applicable to a graph G with an induced subgraph shown on the left, in which vertices a, b, c have degree 3 in G, and moreover, a and b have white neighbours.

▷ Claim 19. Let G' be a graph obtained from G by the butterfly reduction. Then G has a Hamiltonian cycle if and only if G' has a Hamiltonian cycle.

In our algorithm we implement the above rules and reductions whenever they are applicable. We now develop more reductions allowing us to bound the number of vertices in black components. We assume that none of the above rules and reductions is applicable to G.

We omit the proof of the next claim.

▷ Claim 20. Let x be a vertex of degree at least 13. If the neighbourhood of x does not contain two adjacent vertices of degree 3, then G has no Hamiltonian cycle. Otherwise, G has a Hamiltonian cycle if and only if $G - x$ has.

Application of Claim 20 to vertices of large degree either shows that G has no Hamiltonian cycle or reduces the input graph to a graph of maximum degree 12. We will refer to this reduction as the *large degree reduction* and will denote it by (R7).

We omit the proof of the next claim.

▷ Claim 21. The black graph has no induced paths of length 8.

Since graphs of diameter D and maximum degree Δ have fewer than $\frac{\Delta}{\Delta-2}(\Delta-1)^D$ vertices, we conclude that after eliminating vertices of large degree, every black component has fewer than $\frac{12}{10}11^7$ vertices.

To develop more rules and reductions, assume G has a Hamiltonian cycle C. We can further assume that not all vertices of the graph are black, since otherwise the graph contains fewer than $\frac{12}{10}11^7$ vertices, in which case we can solve the problem by brute-force. A sequence of consecutive vertices of C surrounded by white vertices will be called a *black interval*. Observe that each black interval consists of at least two vertices (according to (R4)).

Let K be a black component of G. We will call the vertices of K that have white neighbours the *contact vertices*. Note that K may consists of one or more intervals. Each interval gives rise to exactly two contact vertices. Hence, the number of contact vertices in K is even.

In our next claim, whose proof we omit, we show that for T-subgraph-free graphs, the number of intervals is at most 2.

▷ Claim 22. Any black component consists of at most two intervals.

By Claim 22, if G has a Hamiltonian cycle, then every black component has two or four contact vertices.

(R8) If a black component K has exactly two contact vertices, check if K has a Hamiltonian path connecting the contact vertices. If such a path does not exist, then stop: the input graph has no Hamiltonian cycle. Otherwise, choose arbitrarily a Hamiltonian path connecting the contact vertices, include the edges of the path in the solution and delete all other edges from K.

Rule (R8) destroys black components with two contact vertices, i.e. after its implementation all vertices in such components become white.

Now we discuss the case where each black component has exactly four contact vertices. Let K be such a component with contact vertices v_1, v_2, v_3, v_4. If G has a Hamiltonian cycle, then the vertices of K can be partitioned into two parts each of which forms a path connecting a pair of contact vertices. We will call such a partition a *pairing* (of contact vertices) and will refer to a pairing as the set of edges in the two paths. Also, we will say that two pairings are of the same type, if they pair the contact vertices in the same way. Clearly, if all possible pairings in K have the same type, then it is irrelevant which one to choose, since non-contact vertices of K have no neighbours outside of K.

The above discussion justifies the following two rules.

(R9) If a black component K with four contact vertices does not admit any pairing, then stop: the input graph has no Hamiltonian cycle.

(R10) If in a black component K with four contact vertices all possible pairings have the same type, then choose arbitrarily any such pairing and delete all other edges from K. If this procedure disconnects the graph, then stop: the input graph has no Hamiltonian cycle.

Finally, we analyse the situation when each black component of G admits pairings of at least two different types.

▷ Claim 23. If each black component of the (connected) graph G admits pairings of at least two different types, then G has a Hamiltonian cycle.

Proof. Let K be a black component with contact vertices v_1, v_2, v_3, v_4 and let B and R be two pairings of different types, say B pairs v_1 with v_2 and v_3 with v_4, while R pairs v_1 with v_3 and v_2 with v_4. Assume that

(1) the deletion of all edges of K except for the edges of B disconnects the graph into two components C_{12} (containing vertices v_1 and v_2) and C_{34} (containing vertices v_3 and v_4), and

(2) the deletion of all edges of K except for the edges of R disconnects the graph into two components C_{13} (containing vertices v_1 and v_3) and C_{24} (containing vertices v_2 and v_4).

Note that (1) separates v_1 from v_3 and v_4, while (2) separates v_1 from v_4. Therefore, after the deletion of *all* edges of K vertex v_1 is separated from all other contact vertices. In other words, after the deletion of all edges of K, vertices v_1, v_2, v_3, v_4 belong to pairwise different connected components, say V_1, V_2, V_3, V_4, respectively.

We observe that in each connected component V_i vertex v_i has degree 1 (it is adjacent to a white vertex only). Any other vertex of odd degree in V_i (if there is any) is black, i.e. appears in some black component K'. In the graph $G[K']$ the number of odd vertices is even (by the Handshake lemma). Attaching to $G[K']$ four white neighbours changes the parity of exactly four vertices of K' and hence leaves the number of vertices of K' with odd degrees in the graph G even. Since all vertices of K' belong to only one of the components V_i, we conclude that in each component V_i the number of vertices of odd degree is odd. This is not possible by the Handshake lemma and hence either (1) or (2) is not valid, i.e. we can keep one of the pairings and delete all other edges of K without disconnecting G. This operation destroys K, i.e. makes all vertices of K white.

Applying the above arguments to all black components, one by one, we transform G into a connected graph in which all vertices are white, i.e. to a Hamiltonian cycle. ◁

We summarize the discussion in the following algorithm to solve the problem.

1. Apply rules and reductions (R1) – (R7) as long as they are applicable.
2. If the algorithm did not stop at Step 1 and the graph has fewer than $\frac{12}{10}11^7$ vertices, then solve the problem by brute-force. Otherwise, check the number of contact vertices in black components. If there is a black component with the number of contact vertices different from 2 or 4, then stop: G has no Hamiltonian cycle.
3. If the algorithm did not stop at Step 2, then apply (R8) to black components with two contact vertices, and (R9) and (R10) to black components with four contact vertices.
4. If the algorithm did not stop at Step 3, then find a Hamiltonian cycle according to Claim 23.

Reductions (R8), (R9), (R10) can be implemented in constant time, because the number of vertices in each black component is bounded by a constant. It is also obvious that all other rules, and hence all steps of the algorithm can be implemented in polynomial time. The correctness of the algorithm follows from the proofs of the claims. ◀

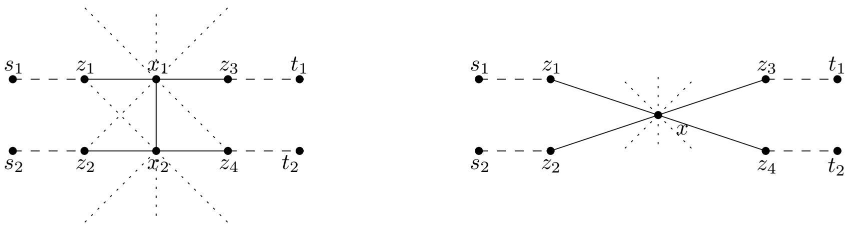


Figure 5 Rule 1. Possible connections in our subgraph (left). What we replace this subgraph with (right). Dotted lines are possible additional edges.

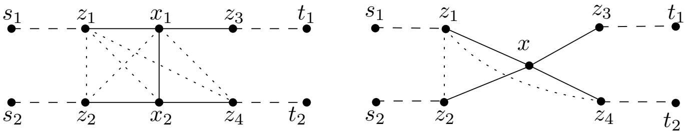


Figure 6 Rule 2. Possible connections in our subgraph (left). What we replace this subgraph with (right). Dotted lines are possible additional edges.

6 k-Induced Disjoint Paths

The case $H = \mathbb{H}_1$ follows from the observation that solutions of k-Induced Disjoint Paths with long paths are solutions of k-Disjoint Paths, which is polynomial-time solvable [30]. We omit the proof details. The case $H = \mathbb{H}_2$ is more involved.

▶ **Theorem 24.** *For all $k \geq 2$, k-*Induced Disjoint Paths *is polynomial-time solvable for $\mathbb{H}_1$-subgraph-free graphs.*

▶ **Theorem 25.** *For all $k \geq 2$, k-*Induced Disjoint Paths *is polynomial-time solvable for $\mathbb{H}_2$-subgraph-free graphs.*

Proof. First, branch on all $O(2^k n^{3k})$ options (so a polynomial number, as k is fixed) of solution paths that have at most three internal vertices. For each branch, we remove the guessed solution paths and the neighbours of the vertices on these paths. Let k still be the number of terminal pairs. We now only look for solution paths with at least four vertices. Branch on all $O(n^{4k})$ options of choosing the first two vertices a_z, b_z on the solution path from every terminal $z \in \{s_i, t_i\}$ for $i \in \{1, \dots, k\}$. In each branch, we remove all other neighbours of z, a_z from the graph, so every terminal z now has degree 1, while a_z has degree 2. We discard the branch if (†) $\{a_z, b_z\} \cap \{a_{z'}, b_{z'}\} \neq \emptyset$ for some terminals z, z' or one of a_z, b_z is the same or neighbours one of $a_{z'}, b_{z'}$ for some terminals z, z' not from the same terminal pair.

We now start a recursive procedure. We first preprocess the input. If b_{s_i} and b_{t_i} are adjacent for some $i \in \{1, \dots, k\}$, then we remove the solution path $s_i, a_{s_i}, b_{s_i}, b_{t_i}, a_{t_i}, t_i$ and their neighbours from the graph. If b_z and $b_{z'}$ are adjacent for some terminals z, z' that do not form a terminal pair, we discard the branch.

We run the polynomial-time algorithm for k-Disjoint Paths from [30] on the remaining terminal pairs. If this results in a no-answer, we discard the branch. Else, we found a solution $P_1, \dots, P_k$. We may assume that each path P_i is induced, or we may shortcut it. If $P_1, \dots, P_k$ is also a solution of k-Induced Disjoint Paths, then we return “yes”. Otherwise, there is (say) an edge (x_1, x_2) between paths $x_1 \in P_1$ and $x_2 \in P_2$. We pick x_1 such that it is closest to t_1 on P_1 and under that condition we pick x_2 such that it is closest to t_2 on P_2.

Let z_1, z_3 be the two neighbours of x_1 on P_1 and z_2, z_4 the two neighbours of x_2 on P_2. We let $S = \{z_1, x_1, z_3, z_2, x_2, z_4\}$. Observe that S contains no terminal by † and the preprocessing. If any $z \in \{z_1, z_2, z_3, z_4\}$ has two neighbours outside of S, then G has a $\mathbb{H}_2$ as a subgraph. Thus we may assume (‡) that each $z \in \{z_1, z_2, z_3, z_4\}$ has at most one neighbour not in S.

By the choice of (x_1, x_2) and as P_1 is induced, z_3 has no neighbours in S except x_1. Suppose the edge (z_1, z_2) exists and one of $\{x_1, x_2\}$ has a neighbour outside of S. Then there is an $\mathbb{H}_2$ with middle path x_1, z_1, z_2 since $s_2 \notin S$. Suppose the edge (z_1, z_4) exists and one of $\{x_1, x_2\}$ has a neighbour outside of S. Then there is an $\mathbb{H}_2$ with middle path x_1, z_1, z_4 since $s_2 \notin S$. Now either the edges (z_1, z_2) and (z_1, z_4) do not exist (see Figure 5), or at least one of them exists and x_1, x_2 have no neighbours outside S (see Figure 6). In the former case, we apply Rule 1, while in the latter case, we apply Rule 2; see Fig. 5 and 6 for their description.

Rule 1 is safe: Suppose that we have a solution to k-Induced Disjoint Paths in G. If this solution uses no vertices in S, then it is already a solution to k-Induced Disjoint Paths in G'. Thus, it must use some vertex in S. If the solution does not use x_1 nor x_2, then recall that by ‡, each of z_1, z_2, z_3, z_4 has at most one neighbour outside of S, and thus the solution must avoid thus S entirely, a contradiction. If the solution uses both x_1 and x_2, then it must use the edge (x_1, x_2). We can substitute the edge (x_1, x_2) in the solution to k-Induced Disjoint Paths in G with x to obtain a solution to k-Induced Disjoint Paths in G'. Hence, without loss of generality, suppose the solution uses x_1. We can substitute this for x to obtain a solution to k-Induced Disjoint Paths in G', unless some other solution path runs through a neighbour q of x_2. Note q cannot be a terminal due to our preprocessing. Hence it has two neighbours p and r on this other solution path, and these are outside of $\{z_1, x_1, z_3\}$ because this path must avoid x_1 and any of its neighbours. But now p, q, r, q, x_2, x_1, z_1, x_1, z_3 forms an $\mathbb{H}_2$ (with middle path q, x_2, x_1), a contradiction.

Suppose we have a solution to k-Induced Disjoint Paths in G'. If this solution does not involve x, then it maps to a solution of k-Induced Disjoint Paths in G. Suppose now it does involve x. Suppose mapping x to either of x_1 or x_2 does not produce a solution to k-Induced Disjoint Paths in G. Then mapping x to either the edge (x_1, x_2) (or the symmetric (x_2, x_1)) must produce a solution to k-Induced Disjoint Paths in G.

Rule 2 is safe: Suppose we have a solution to k-Induced Disjoint Paths in G. If it uses no vertices in S, then it is already a solution to k-Induced Disjoint Paths in G'. Thus, it must use some vertex in S. Suppose the edge (z_1, z_2) exists, and the solution uses (z_1, z_2). Then by ‡, the solution does not use any other vertex from S and we can keep this edge to obtain a solution for k-Induced Disjoint Paths in G'. Suppose the edge (z_1, z_4) exists and the solution uses (z_1, z_4). Then by ‡, the solution does not use any other vertex from S and we can keep this edge to obtain a solution for k-Induced Disjoint Paths in G'.

If the solution uses both x_1 and x_2, then it must use the edge (x_1, x_2), and we can substitute (x_1, x_2) in the solution to k-Induced Disjoint Paths in G with x to obtain a solution to k-Induced Disjoint Paths in G'. Suppose it uses neither x_1 nor x_2. Then by ‡ and the fact that S is used, the solution must use either the edge (z_1, z_4) or (z_1, z_2) and we are in a previous case. Hence, without loss of generality, suppose the solution uses x_1. We can substitute this for x to obtain a solution to k-Induced Disjoint Paths in G'. This is safe, as x_1, x_2 have no neighbours outside S.

Suppose we have a solution to k-Induced Disjoint Paths in G'. If this solution does not involve x then it maps to a solution of k-Induced Disjoint Paths in G. Suppose now it does involve x. Suppose mapping x to either of x_1 or x_2 does not produce a solution to k-Induced Disjoint Paths in G. Then mapping x to either the edge (x_1, x_2) (or the symmetric (x_2, x_1)) must produce a solution to k-Induced Disjoint Paths in G.

Next, we show that any graph G' obtained after applying Rule 1 or 2 is also $\mathbb{H}_2$-subgraph-free. Suppose G' has an $\mathbb{H}_2$. Then this $\mathbb{H}_2$ must contain x. If x is a leaf in $\mathbb{H}_2$, then G already had this $\mathbb{H}_2$ involving either x_1 or x_2. Suppose x is a degree-3 vertex in this $\mathbb{H}_2$. If the neighbours of x in the $\mathbb{H}_2$ were all neighbours of x_1 or all neighbours of x_2 in G, then G already had this $\mathbb{H}_2$, a contradiction. Let z_1' and z_2' be the leafs of the $\mathbb{H}_2$ adjacent to x in G'.

Suppose z_1' and z_2' are both adjacent to x_2 and both not to x_1. Then the middle vertex of the $\mathbb{H}_2$ is only adjacent to x_1. Ideally, we would replace x by x_1, z_1' by z_1 and z_2' by z_3. This does not work if (say) z_1 is part of the $\mathbb{H}_2$. However, z_1' and z_2' are both not z_1, as z_1 is adjacent to x_1, and we would contradict our assumption on the adjacency of z_1' and z_2'. We now consider three cases, depending on where z_1 is in the $\mathbb{H}_2$.

Suppose z_1 is a leaf of the $\mathbb{H}_2$. By ‡ and the inducedness of paths, its neighbouring degree-3 vertex cannot be one of z_2, z_3, z_4. Hence, this must be the unique neighbour p of z_1 outside S. The other neighbours q, r of p on the $\mathbb{H}_2$, where r is the middle vertex, are both not z_3, as P_1 is induced. Hence, q, p, z_1, p, r, x_1, x_2, x_1, z_3 form an $\mathbb{H}_2$, a contradiction.

Suppose that z_1 is the middle vertex of the $\mathbb{H}_2$. By ‡, the other degree-3 vertex of the $\mathbb{H}_2$ cannot be z_2 or z_4, so it must be the unique neighbour p of z_1 outside S. The other neighbours q, r of p on the $\mathbb{H}_2$, which are both leafs of the $\mathbb{H}_2$, are both not z_3 since P_1 is induced. Hence, G has a $\mathbb{H}_2$ formed by q, p, r, p, z_1, x_1, x_2, x_1, z_3, a contradiction.

Suppose that z_1 is a degree-3 vertex of the $\mathbb{H}_2$. Let p be the unique neighbour of z_1 outside S; it is unique by ‡. Then one of p, z_2, z_3 must be the middle vertex of the $\mathbb{H}_2$ and the other two the leafs neighbouring z_1. If the middle vertex is z_2, then z_1', x_2, z_2', x_2, z_2, z_1, p, z_1, z_4 is a $\mathbb{H}_2$ in G, a contradiction. The other cases are similar. This concludes the argument when z_1' and z_2' are both adjacent to x_2 and both not to x_1.

Suppose instead that, say z_1', is adjacent to x_1 and the other, z_2', is adjacent to x_2. Let x', x'', z_1'', z_2'' form the remaining vertices of the $\mathbb{H}_2$ where x, x', x'' and z_1'', x'', z_2'' are both paths of length 2 in this $\mathbb{H}_2$. Thus, z_1', x, z_2', x, x', x'' and z_1'', x'', z_2'' form the $\mathbb{H}_2$ in G'. Without loss of generality, suppose x' was adjacent to x_1 in G. Now it is clear that z_1', x_1, x_2, x_1, x', x'' and z_1'', x'', z_2'' formed an $\mathbb{H}_2$ in G.

Finally, suppose that x is the degree-2 vertex in $\mathbb{H}_2$. Let z_1', x', z_2', x', x, x'', z_1'', x'', z_2'' be the paths that form the $\mathbb{H}_2$ in G'. Suppose, without loss of generality, that x' was adjacent to x_1 in G. If x'' was also adjacent to x_1 in G, then z_1', x', z_2', x', x_1, x'', z_1'', x'', z_2'' are paths that form an $\mathbb{H}_2$ in G. Suppose now that x'' was adjacent to x_2 but not x_1 in G and we may also assume that x' is adjacent to x_1 but not x_2. Now z_1', x', z_2', x', x_1, x_2, z_2, x_2, z_4 are paths that form a $\mathbb{H}_2$ in G, unless $\{z_2, z_4\} \cap \{z_1', z_2'\} \neq \emptyset$. Without loss of generality, suppose $z_2 = z_1'$. Note that $z_2 \neq s_2$ (recall that S does not contain any terminal). Let p be the next vertex on the path from t_2 to s_2 after z_2. Then p, z_2, x_2, z_2, x', x_1, z_1, x_1, z_3 is an $\mathbb{H}_2$ in G (note that $\{z_1, z_3\} \cap \{x', z_2, p\} = \emptyset$), a contradiction.

Finally, note that x_1 and x_2 cannot be z or a_z for some terminal z, as these vertices have degree 1 and 2 respectively, while x_1, x_2 have degree at least 3. Moreover, $\{x_1, x_2\} \neq \{b_z, b_z'\}$ for some terminals z, z' by our preprocessing. Hence, Rules 1 and 2 preserve †.

We can recognize and apply Rules 1 and 2 in polynomial time. This decreases the size of the graph by one vertex and we recurse. Hence, our algorithm runs in polynomial time. ◀

For our next result we follow the proof from Section 2.4 in [24] by carefully p-subdividing *some* of the edges of that construction. We omit the proof details.

▶ **Theorem 26.** *For all $k \geq 2$, k-*Induced Disjoint Paths *is* NP*-complete for subcubic $(\mathbb{H}_4, \ldots, \mathbb{H}_\ell)$-subgraph-free graphs for all $\ell \geq 4$.*

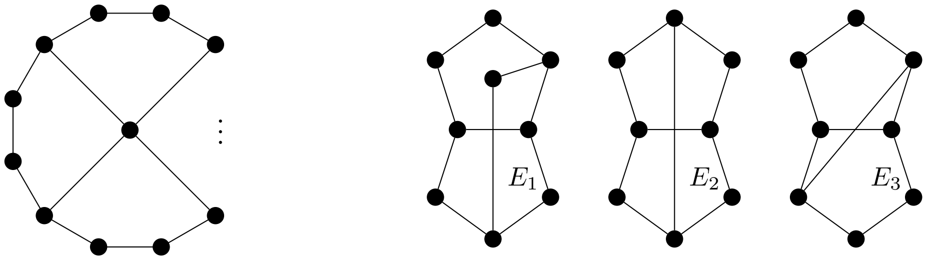


Figure 7 The C_5-flower F_n and the $\mathbb{H}_3$-subgraph-free C_5-critical graphs E_1, E_2 and E_3.

7 C5-Colouring

In this section, we give our polynomial-time certifying algorithm for C_5-Colouring on $\mathbb{H}_3$-subgraph-free graphs. The *C_5-flower F_n* is the graph (see Figure 7) that we get from C_{3n} (for $n \geq 3$) by adding a new central vertex with an edge to every third vertex of C_{3n}. If n is odd, we call F_n an *odd C_5-flower*, and if it is even we call F_n an *even C_5-flower*. We refer to the graphs E_1, E_2 and E_3 shown in Figure 7 as *exceptional graphs*.

The following lemma (whose proof is a simple exercise) shows that all these graphs are *C_5-critical*, that is, they are not C_5-colourable but every proper subgraph of them is.

▶ **Lemma 27.** *The graph K_3, the odd flowers F_n for odd $n \geq 3$ and the exceptional graphs E_1, E_2 and E_3 are all $\mathbb{H}_3$-subgraph-free and C_5-critical.*

We can now show a structural result, which we use to prove our algorithmic result. We omit its proof.

▶ **Theorem 28.** *The only $\mathbb{H}_3$-subgraph-free C_5-critical graphs are K_3, odd flowers F_n ($n \geq 3$) and exceptional graphs E_1, E_2, E_3. Equivalently, the following statements all hold:*

1. *All $\mathbb{H}_3$-subgraph-free graphs of girth at least 6 are C_5-colourable.*
2. *The only $\mathbb{H}_3$-subgraph-free C_5-critical graphs of girth 5 are E_1, E_2 and odd C_5-flowers F_n.*
3. *The only $\mathbb{H}_3$-subgraph-free C_5-critical graph of girth 4 is E_3.*

▶ **Theorem 29.** *There exists a polynomial-time certifying algorithm for C_5-*Colouring *on $\mathbb{H}_3$-subgraph-free graphs.*

Proof. As every graph that does not map to C_5 must contain a C_5-critical subgraph, it suffices, due to Theorem 28, to detect the non-existence of the graphs K_3, E_1, E_2, E_3 and F_n (odd $n \geq 3$) in a $\mathbb{H}_3$-subgraph-free graph G. For the graphs K_3, E_1, E_2 and E_3 we can simply use brute force. To detect an odd C_5-flower F_n in polynomial time, we observe that for a fixed centre vertex, v_0 we can make an auxiliary graph on its neighbours putting an edge between two if there is a path on three edges between them in G. Now, G contains an odd C_5-flower with centre v_0 if and only if this auxiliary graph has an odd cycle. We can check this in polynomial time for each v_0, so can find an odd C_5-flower in G polynomial time. ◀

8 Conclusions

We took four classic problems, Hamilton Cycle, k-Induced Disjoint Paths, C_5-Colouring and Star 3-Colouring, that are "easy" on bounded treewidth, but for which we showed that their hardness on subcubic graphs is not preserved under edge subdivision. We gave polynomial and NP-completeness results for $\mathcal{H}$-subgraph-free graphs when $\mathcal{H}$ is some subset of $\{\mathbb{H}_1, \mathbb{H}_2, \ldots\}$, but we need to better understand the case $\mathcal{H} = \{\mathbb{H}_i\}$ $(i \geq 1)$.

▶ **Open Problem 1.** *Is there a graph $\mathbb{H}_\ell$ such that* Hamilton Cycle *is* NP*-complete for $\mathbb{H}_\ell$-subgraph-free graphs?*

We note that the case $\mathbb{H}_3$ is the only missing case for obtaining a dichotomy for k-Induced Disjoint Paths on $\mathbb{H}_i$-subgraph-free graphs,

▶ **Open Problem 2.** *What is the complexity of* k-Induced Disjoint Paths *for $\mathbb{H}_3$-subgraph-free graphs?*

If C_5-Colouring on $\mathbb{H}_i$-subgraph-free graphs is polynomial-time solvable when $i = 0 \bmod 3$, then we would get a dichotomy for C_5-Colouring on $\mathbb{H}_i$-subgraph-free graphs based on $i \bmod 3$.

▶ **Open Problem 3.** *What is the complexity of* C_5-Colouring *for $\mathbb{H}_i$-subgraph-free graphs, when $i = 0 \bmod 3$?*

If Star 3-Colouring on $\mathbb{H}_{2i}$-subgraph-free graphs is polynomial-time solvable for $i \geq 1$, then we would get a dichotomy for Star 3-Colouring on $\mathbb{H}_i$-subgraph-free graphs based on $i \bmod 2$.

▶ **Open Problem 4.** *What is the complexity of* Star 3-Colouring *for $\mathbb{H}_{2i}$-subgraph-free graphs for $i \geq 1$?*

Moreover, even though Star k-Colouring is not C2 for $k \geq 10$ (Proposition 14), this is not known for $4 \leq k \leq 9$. In particular, Shalu and Antony asked about the case $k = 4$ in [31], and we recall their open problem.

▶ **Open Problem 5.** *What is the complexity of* Star 4-Colouring *for subcubic graphs?*

We also still need to determine whether the C12-problems k-Induced Disjoint Paths and C_5-Colouring are even C12' just like Hamilton Cycle and Star 3-Colouring. In order to know this, we must solve the following two problems.

▶ **Open Problem 6.** *What is the complexity of* k-Induced Disjoint Paths *for subcubic graphs of girth g for $g \geq 3$?*

▶ **Open Problem 7.** *What is the complexity of* C_5-Colouring *for subcubic graphs of girth g for $g \geq 3$?*

We also do not know the complexity of k-Induced Disjoint Paths, for $k \geq 2$, on graphs of girth at least g with an additional degree bound, whereas the best degree bound for C_5-Colouring is $6 \cdot 5^{13}$. Namely, for every $g \geq 3$, C_5-Colouring is NP-complete for graphs with girth at least g and with maximum degree at most $6 \cdot 5^{13}$ (Theorem 6).

Finally, there exist other problems that are NP-complete for bipartite graphs in which one partition class has maximum degree 2 and thus on $(\mathbb{H}_1, \mathbb{H}_3, \ldots)$-subgraph-free graphs. One example of such a problem is Matching Cut [27]. Another example is Acyclic 3-Colouring, for which we can show the same results as for Star 3-Colouring in Theorem 5 by using the same arguments. However, in contrast to Star 3-Colouring, we do not know if Acyclic 3-Colouring satisfies C2 and we recall the following open problem from Shalu and Antony [32].

▶ **Open Problem 8.** *What is the complexity of* Acyclic 3-Colouring *for subcubic graphs?*

References

1 Michael O. Albertson, Glenn G. Chappell, Henry A. Kierstead, André Kündgen, and Radhika Ramamurthi. Coloring with no 2-colored P_4's. *Electronic Journal of Combinatorics*, 11, 2004.

2 Vladimir E. Alekseev, Rodica Boliac, Dmitry V. Korobitsyn, and Vadim V. Lozin. NP-hard graph problems and boundary classes of graphs. *Theoretical Computer Science*, 389:219–236, 2007. doi:10.1016/J.TCS.2007.09.013.

3 Vladimir E. Alekseev and Dmitry V. Korobitsyn. Complexity of some problems on hereditary graph classes. *Diskretnaya Matematika*, 4:34–40, 1992.

4 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

5 Daniel Bienstock, Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a forest. *Journal of Combinatoral Theory, Series B*, 52:274–283, 1991. doi:10.1016/0095-8956(91)90068-U.

6 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82:3566–3587, 2020. doi:10.1007/S00453-020-00737-Z.

7 Hans L. Bodlaender, Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs IV: The Steiner Forest problem. *Proc. IWOCA 2024, LNCS*, 14764:206–217, 2024. doi:10.1007/978-3-031-63021-7_16.

8 Rodica Boliac and Vadim V. Lozin. On the clique-width of graphs in hereditary classes. *Proc. ISAAC 2022, LNCS*, 2518:44–54, 2002. doi:10.1007/3-540-36136-7_5.

9 Maria Chudnovsky, Shenwei Huang, Pawe ł Rzążewsk, Sophie Spirkl, and Mingxian Zhong. Complexity of C_k-coloring in hereditary classes of graphs. *Proc. ESA 2019, LIPIcs*, 144:31:1–31:15, 2019. doi:10.4230/LIPICS.ESA.2019.31.

10 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

11 David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30:289–293, 1980. doi:10.1016/0012-365X(80)90236-8.

12 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989. doi:10.1016/0004-3702(89)90037-4.

13 Tala Eagling-Vose, Barnaby Martin, Daniël Paulusma, and Siani Smith. Graph homomorphism, monotone classes and bounded pathwidth. *Proc. CiE 2024, LNCS*, 14773:233–251, 2024. doi:10.1007/978-3-031-64309-5_19.

14 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability & Computing*, 7:375–386, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=46667.

15 Anna Galluccio, Pavol Hell, and Jaroslav Nešetřil. The complexity of H-colouring of bounded degree graphs. *Discrete Mathematics*, 222:101–109, 2000. doi:10.1016/S0012-365X(00)00009-1.

16 Alfred Geroldinger and Imre Z. Ruzsa. *Combinatorial Number Theory and Additive Group Theory*. Birkhäuser, 2009.

17 Petr A. Golovach and Daniël Paulusma. List coloring in the absence of two subgraphs. *Discrete Applied Mathematics*, 166:123–130, 2014. doi:10.1016/J.DAM.2013.10.010.

18 Petr A. Golovach, Daniël Paulusma, and Bernard Ries. Coloring graphs characterized by a forbidden subgraph. *Discrete Applied Mathematics*, 180:101–110, 2015. doi:10.1016/J.DAM.2014.08.008.

19 Pavol Hell and Jaroslav Nesetril. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

20 Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs I: The framework. *CoRR*, 2211.12887, 2022.

21 Matthew Johnson, Barnaby Martin, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs III: When problems are tractable on subcubic graphs. *Proc. MFCS 2024, LIPIcs*, 272:57:1–57:15, 2023. doi:10.4230/LIPICS.MFCS.2023.57.

22 Marcin Kamiński. Max-Cut and containment relations in graphs. *Theoretical Computer Science*, 438:89–95, 2012. doi:10.1016/J.TCS.2012.02.036.

23 Nicholas Korpelainen, Vadim V. Lozin, Dmitriy S. Malyshev, and Alexander Tiskin. Boundary properties of graphs for algorithmic graph problems. *Theoretical Computer Science*, 412:3545–3554, 2011. doi:10.1016/J.TCS.2011.03.001.

24 Benjamin Lévêque, David Y. Lin, Frédéric Maffray, and Nicolas Trotignon. Detecting induced subgraphs. *Discrete Applied Mathematics*, 157:3540–3551, 2009. doi:10.1016/J.DAM.2009.02.015.

25 Vadim V. Lozin. The hamiltonian cycle problem and monotone classes. *Proceedings of IWOCA 2024, LNCS*, 14764:460–471, 2024. doi:10.1007/978-3-031-63021-7_35.

26 Vadim V. Lozin and Igor Razgon. Tree-width dichotomy. *European Journal of Combinatorics*, 103:103517, 2022. doi:10.1016/J.EJC.2022.103517.

27 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13:527–536, 1989. doi:10.1002/JGT.3190130502.

28 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

29 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41:92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

30 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The Disjoint Paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995. doi:10.1006/JCTB.1995.1006.

31 M. A. Shalu and Cyriac Antony. Star colouring of bounded degree graphs and regular graphs. *Discrete Mathematics*, 345:112850, 2022. doi:10.1016/J.DISC.2022.112850.

32 M. A. Shalu and Cyriac Antony. Hardness transitions and uniqueness of acyclic colouring. *Discrete Applied Mathematics*, 345:77–98, 2024. doi:10.1016/J.DAM.2023.11.030.

33 Yossi Shiloach. A polynomial solution to the undirected Two Paths problem. *Journal of the ACM*, 27:445–456, 1980. doi:10.1145/322203.322207.

Complexity of Local Search for Euclidean Clustering Problems

Bodo Manthey
Faculty of Electrical Engineering, Mathematics, and Computer Science,
University of Twente, The Netherlands

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Jesse van Rhijn
Faculty of Electrical Engineering, Mathematics, and Computer Science,
University of Twente, The Netherlands

Frank Sommer
Institute of Logic and Computation, TU Wien, Austria

Abstract

We show that the simplest local search heuristics for two natural Euclidean clustering problems are PLS-hard. First, we show that the Hartigan–Wong method, which is essentially the FLIP heuristic, for k-MEANS clustering is PLS-hard, even when $k = 2$. Second, we show the same result for the FLIP heuristic for MAX CUT, even when the edge weights are given by the (squared) Euclidean distances between the points in some set $\mathcal{X} \subseteq \mathbb{R}^d$; a problem which is equivalent to MIN SUM 2-CLUSTERING.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization; Theory of computation → Facility location and clustering

Keywords and phrases Local search, PLS-complete, max cut, k-means, partitioning problem, flip-neighborhood

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.48

Related Version *Full Version*: https://arxiv.org/abs/2312.14916

Funding *Jesse van Rhijn*: Supported by NWO grant OCENW.KLEIN.176.
Frank Sommer: Supported by the Alexander von Humboldt Foundation.

1 Introduction

Clustering problems arise frequently in various fields of application. In these problems, one is given a set of objects, often represented as points in $\mathbb{R}^d$, and is asked to partition the set into *clusters*, such that the objects within a cluster are similar to one another by some measure. For points in $\mathbb{R}^d$, a natural measure is the (squared) Euclidean distance between two objects. In this paper, we consider two Euclidean clustering problems that use this similarity measure: k-MEANS clustering and SQUARED EUCLIDEAN MAX CUT.

k-Means. One well-studied clustering problem is k-MEANS [10, 23]. In this problem, one is given a set of points $\mathcal{X} \subseteq \mathbb{R}^d$ and an integer k. The goal is to partition $\mathcal{X}$ into exactly k clusters such that the total squared distance of each point to the centroid of its cluster is minimized. Formally, one seeks to minimize the clustering cost

$$\sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mathrm{cm}(C_i)\|^2 \quad \text{where} \quad \mathrm{cm}(C_i) = \frac{1}{|C_i|} \sum_{x \in C_i} x.$$

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 48; pp. 48:1–48:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Being NP-hard even when $k = 2$ [3] or when $\mathcal{X} \subseteq \mathbb{R}^2$ [31], k-Means has been extensively studied from the perspective of approximation algorithms [8, 21, 26, 34]. Nevertheless, local search remains the method of choice for practitioners [10, 23].

The most well-known local search algorithm for k-Means is Lloyd's method [30]. Here, one alternates between two steps in each iteration. In the first step, each point is assigned to its closest cluster center, and in the second step the cluster centers are recalculated from the newly formed clusters.

This algorithm was shown to have worst-case super-polynomial running time by Arthur and Vassilvitskii [7], with the result later improved to exponential running time even in the plane by Vattani [45]. Moreover, Roughgarden and Wang [39] showed it can implicitly solve PSPACE-complete problems. On the other hand, Arthur et al. [6] proved that Lloyd's method has smoothed polynomial running time on Gaussian-perturbed point sets, providing a degree of explanation for its effectiveness in practice.

Recently Telgarsky and Vattani [44] revived interest in another, older local search method for k-Means due to Hartigan and Wong [20]. This algorithm, the Hartigan–Wong method, is much simpler: one searches for a single point that can be reassigned to some other cluster for a strict improvement in the clustering cost. In other words, the Hartigan–Wong method is the Flip heuristic. In the following, we always use Flip instead of Hartigan–Wong to indicate that this is the most simple heuristic for this problem and to keep the name for the used heuristic consistent. Despite this simplicity, Telgarsky and Vattani [44] show that the Flip heuristic is more powerful than Lloyd's method, in the sense that the former can sometimes improve clusterings produced by the latter, while the converse does not hold.

A similar construction to that of Vattani for Lloyd's method shows that there exist instances on which the Flip heuristic can take exponentially many iterations to find a local optimum, even when all points lie on a line [33]. However, this example follows a contrived sequence of iterations. Moreover, k-Means can be solved optimally for instances in which all points lie on a line. Thus, the question remains whether stronger worst-case examples exist, and what the complexity of finding locally optimal clusterings is.

Squared Euclidean Max Cut. Another clustering problem similar to k-Means is Squared Euclidean Max Cut. Recall that Max Cut asks for a subset of vertices S of a weighted graph $G = (V, E)$, such that the total weight of the edges with one endpoint in S and one in $V \setminus S$ is maximized. This problem emerges in numerous applications, from graph clustering to circuit design to statistical physics [9, 12].

In Squared Euclidean Max Cut, one identifies the vertices of G with a set $\mathcal{X} \subseteq \mathbb{R}^d$, and assigns each edge a weight equal to the squared Euclidean distance between its endpoints. This problem is equivalent to Min Sum 2-Clustering (although not in approximation), where one seeks to minimize

$$\sum_{x,y \in X} \|x - y\|^2 + \sum_{x,y \in Y} \|x - y\|^2$$

over all partitions (X, Y) of $\mathcal{X}$. Also this special case of Max Cut is NP-hard [2]. In a clustering context, the problem was studied by Schulman [42] and Hasegawa et al. [21], leading to exact and approximation algorithms.

Given the computational hardness of Max Cut, practitioners often turn to heuristics. Some of the resulting algorithms are very successful, such as the Kernighan-Lin heuristic [27] and the Fiduccia-Mattheyses algorithm [19]. Johnson et al. [24] note that the simple Flip heuristic, where one moves a single vertex from one side of the cut to the other, tends to converge quickly in practice. Schäffer and Yannakakis [40] later showed that it has exponential running time in the worst case.

One may wonder whether Flip performs better for Squared Euclidean Max Cut. Etscheid and Röglin [17, 16] performed a smoothed analysis of Flip in this context, showing a smoothed running time of $2^{O(d)} \cdot \text{poly}(n, 1/\sigma)$ for Gaussian-perturbed instances, where σ denotes the standard deviation of the Gaussian noise. On the other hand, they also exhibited an instance in $\mathbb{R}^2$ on which there exists an exponential-length improving sequence of iterations, with the caveat that not all edges are present in the instance. Like for k-Means, one may ask whether stronger examples exist (e.g. on complete graphs), and what the complexity of finding Flip-optimal solutions is.

Complexity of Local Search. The existence of instances with worst-case exponential running time is common for local search heuristics. To investigate this phenomenon, and local search heuristics in general, Johnson et al. [24] defined a complexity class PLS, for polynomial local search. The class is designed to capture the properties of commonly used local search heuristics and contains pairs consisting of an optimization problem P and a local search heuristic $\mathcal{N}$. In the following we denote such a pair as P/$\mathcal{N}$. PLS-complete problems have the property that their natural local search algorithms have worst-case exponential running time. Johnson et al. [24] showed that the Kernighan-Lin heuristic for the Max Bisection problem (a variant of Max Cut, where the parts of the partition must be of equal size) is PLS-complete. This stands in contrast to the empirical success of this algorithm [27].

Building on this work, Schäffer and Yannakakis [40] later proved that a host of very simple local search heuristics are PLS-complete, including the Flip heuristic for Max Cut. This refuted a conjecture by Johnson et al., who doubted that such simple heuristics could be PLS-complete. Elsässer and Tscheuschner [15] later showed that this remains true even in the very restricted variant where the input graph has maximum degree five, which we will refer to as Max Cut-5.

Schäffer and Yannakakis defined a new type of PLS-reduction called a *tight* reduction. In addition to showing completeness for PLS, this type of reduction also transfers stronger properties on the running time of local search heuristics between PLS problems.

Since the introduction of PLS, many local search problems have been shown to be PLS-complete, including such successful heuristics as Lin-Kernighan's algorithm for the TSP [38] or the k-Swap-neighborhood heuristic for Weighted Independent Set [28] for $k \geq 3$. For a non-exhaustive list, see Michiels, Korst and Aarts [36, Appendix C].

Our Contribution. Given the existence of k-Means instances where the Flip heuristic has worst-case exponential running time, one may ask whether this heuristic is PLS-hard. In this work, we answer this question in the affirmative.

▶ **Theorem 1.1.** *For each $k \geq 2$, k-Means/Flip is* PLS*-hard.*

Just as with k-Means/Flip, we ask whether Squared Euclidean Max Cut with the Flip heuristic is PLS-hard. Again, we answer this question affirmatively. In addition, we show the same result for Euclidean Max Cut, where the distances between the points are not squared.

▶ **Theorem 1.2.** *Euclidean Max Cut/Flip and Squared Euclidean Max Cut/Flip are* PLS*-hard.*

We note that PLS-hardness results for Euclidean local optimization problems are rather uncommon. We are only aware of one earlier result by Brauer [13], who proved PLS-completeness of a local search heuristic for a discrete variant of k-Means. This variant chooses k cluster centers among the set of input points, after which points are assigned to their closest center. The heuristic they consider removes one point from the set of cluster centers and adds another. In their approach, they first construct a metric instance, and then show that this instance can be embedded into $\mathbb{R}^d$ using a theorem by Schoenberg [41]. In contrast, we directly construct instances in $\mathbb{R}^d$.

In addition to showing PLS-hardness of k-Means/Flip and Squared Euclidean Max Cut/Flip, we also show that there exist specific hard instances of these problems, as well as all other problems considered in this paper.

▶ **Theorem 1.3.** *Each local search problem L considered in this work (see Section 2.2) fulfills the following properties:*

1. *L is* PLS*-hard.*
2. *For each n, one can compute in polynomial time an instance of L of size $n^{\mathcal{O}(1)}$ with an initial solution that is exponentially far away from any local optimum.*
3. *The problem of computing the locally optimal solution obtained from performing a standard local search algorithm based on the neighborhood of L is* PSPACE*-hard for L.*

A formal definition of Property 3 is given in Section 2.1. In particular, this result shows that there exists an instance of k-Means such that there exists an initial solution to this instance that is exponentially many iterations away from *all* local optima [40]. By contrast, the earlier result [33] only exhibits an instance with a starting solution that is exponentially many iterations away from *some* local optimum. Moreover, Theorem 1.3 yields instances where Flip has exponential running time in Squared Euclidean Max Cut on *complete* geometric graphs, unlike the older construction which omitted some edges [16].

2 Preliminaries and Notation

Throughout this paper, we will consider all graphs to be undirected unless explicitly stated otherwise. Let $G = (V, E)$ be a graph. For $v \in V$, we denote by $d(v)$ the *degree* of v in G, and by $N(v)$ the set of *neighbors* of v.

Let $S, T \subseteq V$. We write $E(S, T)$ for the set of edges with one endpoint in S and one endpoint in T. For $S \subseteq V$, we write $\delta(S) = E(S, V \setminus S)$ for the *cut induced by* S. We will also refer to the partition $(S, V \setminus S)$ as a *cut*; which meaning we intend will be clear from the context. If $|\,|S| - |V \setminus S|\,| \leq 1$, we will call the cut $(S, V \setminus S)$ a *bisection.* Given $v \in V$ we will also write $\delta(v) = \delta(\{v\})$, which is the set of *edges incident to* v.

Let $F \subseteq E$. Given a function $f : E \to \mathbb{R}$, we denote by $f(F) = \sum_{e \in F} f(e)$ the *total value* of f on the set of edges F. If $F = E(X, Y)$ for some sets $X, Y \subseteq V$, we will abuse notation to write $f(X, Y) = f(F)$.

2.1 The Class PLS

For convenience, we summarize the formal definitions of local search problems and the associated complexity class PLS, as devised by Johnson et al. [24].

A *local search problem* P is defined by a set of instances I, a set of feasible solutions $F_I(x)$ for each instance $x \in I$, a *cost function* c that maps pairs of a solution of $F_I(x)$ and an instance x to $\mathbb{Z}$, and a *neighborhood function* $\mathcal{N}$ that maps a solution of $F_I(x)$ and an instance x to a subset of $F_I(x)$. Typically, the neighborhood function is constructed so that it is easy to compute some $s' \in \mathcal{N}(s, x)$ for any given $s \in F_I(x)$.

This characterization of local search problems gives rise to the *transition graph* defined by an instance of such a problem.

▶ **Definition 2.1.** *Given an instance $x \in I$ of a local search problem P, we define the* transition graph *$T(x)$ as the directed graph with vertex set $F_I(x)$, with an edge from s to s' if and only if $s' \in \mathcal{N}(s, x)$ and $c(s, x) < c(s', x)$ (assuming P is a maximization problem; otherwise, we reverse the inequality). The* height *of a vertex s in $T(x)$ is the length of the shortest path from s to a sink of $T(x)$.*

The class PLS is defined to capture the properties of local search problems that typically arise in practical applications. Formally, P is contained in PLS if the following are all true:

1. There exists a deterministic polynomial-time algorithm A that, given an instance $x \in I$, computes some solution $s \in F_I(x)$.
2. There exists a deterministic polynomial-time algorithm B that, given $x \in I$ and $s \in F_I(x)$, computes the value of $c(s, x)$.
3. There exists a deterministic polynomial-time algorithm C that, given $x \in I$ and $s \in F_I(x)$, either computes a solution $s' \in \mathcal{N}(s, x)$ with $c(s', x) > c(s, x)$ (in the case of a maximization problem), or outputs that such a solution does not exist.

Intuitively, algorithm A gives us some initial solution from which to start an optimization process, algorithm B ensures that we can evaluate the quality of solutions efficiently, and algorithm C drives the local optimization process by either determining that a solution is locally optimal or otherwise giving us an improving neighbor. Based on the algorithms A and C, one can define the "standard algorithm problem" for P as follows.

▶ **Definition 2.2.** *Let P be a local search problem and let I be an instance of P. Moreover, let $s^*(I)$ be the unique local optimum obtained by starting with the solution outputted by algorithm A and replacing the current solution by the better solution outputted by C, until reaching a local optimum. The* standard algorithm problem *for P asks for a given instance I of P and a locally optimal solution s' for I with respect to $\mathcal{N}$, whether s' is exactly the solution $s^*(I)$.*

It was shown that for many local search problems the standard algorithm problem is PSPACE-complete [13, 36, 37, 40].

Given problems $P, Q \in$ PLS, we say that P is PLS-*reducible* to Q (written $P \leq_{\mathsf{PLS}} Q$) if the following is true.

1. There exist polynomial-time computable functions f, g, such that f maps instances x of P to instances $f(x)$ of Q, and g maps pairs (solution s of $f(x), x$) to feasible solutions of P.
2. If a solution s of $f(x)$ is locally optimal for $f(x)$, then $g(s, x)$ is locally optimal for x.

The idea is that, if $Q \in$ PLS is efficiently solvable, then P is also efficiently solvable: simply convert an instance of P to Q using f, solve Q, and convert the resulting solution back to a solution of P using g. As usual in complexity theory, if P is *complete* for PLS and $P \leq_{\mathsf{PLS}} Q$, then Q is also complete for PLS.

In addition to this standard notion of a PLS-reduction, Schäffer and Yannakakis [40] defined so-called *tight* reductions. Given PLS problems P and Q and a PLS-reduction (f, g) from P to Q, the reduction is called *tight* if for any instance x of P we can choose a subset $\mathcal{R}$ of the feasible solutions of $f(x)$ of Q such that:

1. $\mathcal{R}$ contains all local optima of $f(x)$.
2. For every feasible solution s of x, we can construct a feasible solution $q \in \mathcal{R}$ of $f(x)$ such that $g(q, x) = s$.
3. Suppose the transition graph $T(f(x))$ of $f(x)$ contains a directed path from s to s' such that $s, s' \in \mathcal{R}$, but all internal vertices lie outside of $\mathcal{R}$, and let $q = g(s, x)$ and $q' = g(s', x)$. Then either $q = q'$, or the transition graph $T(x)$ of x contains an edge from q to q'.

The set $\mathcal{R}$ is typically called the set of *reasonable solutions to* $f(x)$. Here, the intuition is that tight reductions make sure that the height of a vertex s of $T(f(x))$ is not smaller than that of $g(s,x)$ in $T(x)$. Note that a reduction (f,g) is trivially tight if $T(f(x))$ is isomorphic to $T(x)$.

Tight PLS-reductions have two desired properties [1, Chapter 2]. Suppose P reduces to Q via a tight reduction. First, if the standard algorithm problem for P is PSPACE-complete, then the standard algorithm problem for Q is PSPACE-complete as well. Second, if there exists an instance x of P such that there exists a solution of x that is exponentially far away from any local optimum, then such an instance exists for Q as well. Note that this first property holds irrespective of the choices made by the algorithm C for Q [40].

2.2 Definitions of Local Search Problems

We will be concerned with various local search problems. In the following we provide a summary of the problems that appear in this paper, and provide definitions for each. Some of the problems considered in this paper are not the most natural ones, but we need them as intermediate problems for our reductions. Moreover, these problems might be useful to show PLS-hardness of other problems having cardinality constraints.

Before introducing the problems themselves, we first provide a more abstract view of the problems, since they have many important aspects in common. Each problem in the list below is a type of partitioning problem, where we are given a finite set S and are asked to find the "best" partition of S into k sets (indeed, for all but one problem, we have $k = 2$). What determines whether some partition is better than another varies; this is determined by the cost function of the problem in question.

▶ **Definition 2.3.** *Given a partition $\mathcal{P} = \{S_1, \dots, S_k\}$ of S, a partition $\mathcal{P}'$ is a neighbor of $\mathcal{P}$ in the* Flip neighborhood *if $\mathcal{P}'$ can be obtained by moving exactly one element from some $S_i \in \mathcal{P}$ to some other $S_j \in \mathcal{P}$. In other words, if $\mathcal{P}' = \{S_1, \dots, S_i', \dots, S_j', \dots, S_k\}$ where for some $v \in S_i$ we have $S_i' = S_i \setminus \{v\}$ and $S_j' = S_j \cup \{v\}$.*

The Flip neighborhood as defined above is perhaps the simplest neighborhood structure for a partitioning problem. For each problem in the list below, we consider only the Flip neighborhood in this paper. Recall that the Flip heuristic for k-Means is also referred to as the Hartigan–Wong method [20].

Max Cut

Input: A graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{Z}_{\geq 0}$.

Output: A partition (X, Y) of V such that $w(X, Y)$ is maximal.

We will mainly be concerned with several variants of Max Cut. For some fixed integer d, by Max Cut-d we denote the restriction of the problem to graphs with maximum degree d. In Densest Cut, one aims to maximize $\frac{w(X,Y)}{|X|\cdot|Y|}$ rather than just $w(X,Y)$. The minimization version of this problem is called Sparsest Cut. The problem Odd Max Bisection is identical to Max Cut, with the added restrictions that the number of vertices must be odd and that the two halves of the cut differ in size by exactly one. The minimization version of the problem is called Odd Min Bisection.

The definitions of Odd Max/Min Bisection are somewhat unconventional, as one usually considers these problem with an even number of vertices and with the Swap neighborhood, where two solutions are neighbors if one can be obtained from the other by swapping a pair of vertices between parts of the partition. Hardness of Max Bisection/Swap was shown by Schäffer and Yannakakis [40] in a short reduction from Max Cut/Flip, providing as a corollary also a simple hardness proof for the Kernighan-Lin heuristic [27]. The reason we require the Flip neighborhood is that we aim to reduce this problem to Squared Euclidean Max Cut/Flip, where we run into trouble when we use the Swap neighborhood (see Section 3 for details).

Squared Euclidean Max Cut

Input: A set of n points $\mathcal{X} \subseteq \mathbb{R}^d$.

Output: A partition (X, Y) of $\mathcal{X}$ such that $\sum_{x \in X} \sum_{y \in Y} \|x - y\|^2$ is maximal.

Euclidean Max Cut is defined similarly; the only difference is that the actual distances between points enter the objective function, rather than the squared distances.

k-Means

Input: A set of n points $\mathcal{X} \subseteq \mathbb{R}^d$ and an integer $k \geq 2$.

Output: A partition $(C_1, \dots, C_k)$ of $\mathcal{X}$ such that $\sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mathrm{cm}(C_i)\|^2$ is minimal.

The sets $\{C_1, \dots, C_k\}$ are called *clusters*. Note that in this formulation, both k-Means/Flip and Squared Euclidean Max Cut/Flip are not contained in PLS, as their cost functions can take non-integer values. However, we still obtain PLS-hardness for each of these problems, and the existence of specific hard instances (cf. Theorem 1.3). Moreover, this hardness still holds for restricted versions of the problems which do belong to PLS. More technical details are given in the full version.

Positive Not-All-Equal k-Satisfiability (Pos NAE k-SAT)

Input: A boolean formula with clauses of the form $\mathrm{NAE}(x_1, \dots, x_\ell)$ with $\ell \leq k$, where each clause is satisfied if its constituents, all of which are positive, are neither all true nor all false. Each clause C has a weight $w(C) \in \mathbb{Z}$.

Output: A truth assignment of the variables such that the sum of the weights of the satisfied clauses is maximized.

In Odd Half Pos NAE k-SAT, additionally the number of variables is odd and it is required that the number of `true` variables and the number of `false` variables in any solution differ by exactly one. This is analogous to the relationship between Max Cut and Odd Max Bisection.

2.3 Strategy

Both Squared Euclidean Max Cut and k-Means are NP-hard [2, 3]. The reductions used to prove this are quite similar, and can be straightforwardly adapted into PLS-reductions: In the case of Squared Euclidean Max Cut/Flip, we obtain a reduction from Odd Min Bisection/Flip, while for k-Means/Flip we obtain a reduction from Densest Cut/Flip. The latter reduction even works for $k = 2$. These results are given in Lemma 4.3 (k-Means), and Lemmas 4.5 and 4.6 ((Squared) Euclidean Max Cut).

What remains then is to show that the problems we reduce from are also PLS-complete, which takes up the bulk of the work. Figure 1 shows the reduction paths we use.

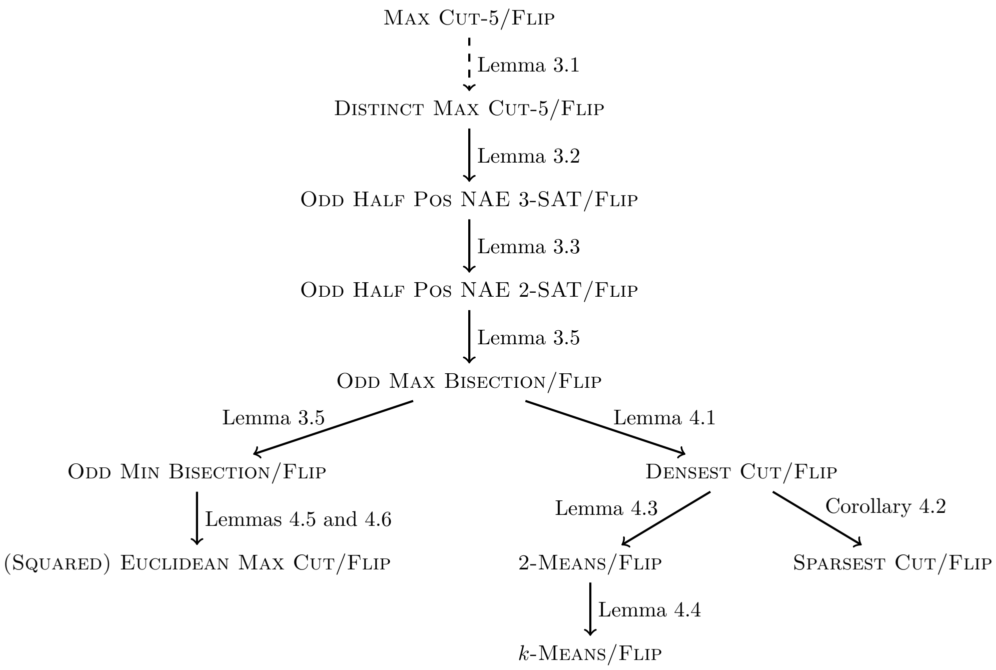


Figure 1 Graph of the PLS-reductions used in this paper. Reductions represented by solid lines are tight, reductions represented by dashed lines are not.

The starting point will be the PLS-completeness of Max Cut-5/Flip, which was shown by Elsässer and Tscheuschner [15]. An obvious next step might then be to reduce from this problem to Max Bisection/Swap, and then further reduce to Odd Max Bisection/Flip. Unfortunately, this turns out to be rather difficult, as the extra power afforded by the Swap neighborhood is not so easily reduced back to the Flip neighborhood. Using this strategy, we can only obtain PLS-hardness of the 2-Flip neighborhood for Squared Euclidean Max Cut, where two points may flip in a single iteration.

We thus take a detour through Odd Half Pos NAE 3-SAT/Flip in Lemma 3.2, which then reduces down to Odd Half Pos NAE 2-SAT/Flip in Lemma 3.3 and finally to Odd Max Bisection/Flip in Lemma 3.5, using a reduction by Schäffer and Yannakakis [40].

From this point, hardness of Squared Euclidean Max Cut/Flip (and with a little extra work, Euclidean Max Cut/Flip) is easily obtained. For k-Means/Flip, we need some more effort, as we still need to show hardness of Densest Cut/Flip. Luckily, this can be done by reducing from Odd Max Bisection/Flip as well, as proved in Lemma 4.1.

Due to space constraints, our proofs are deferred to the full version.

3 Reduction to Odd Min/Max Bisection

The goal of this section is to obtain PLS-completeness of Odd Min/Max Bisection/Flip, from which we can reduce further to our target problems; see Figure 1. We will first construct a PLS-reduction from Max Cut-5/Flip to Odd Half Pos NAE 3-SAT/Flip in Lemma 3.2.

A subtlety is that the reduction only works when we assume that the Max Cut-5 instance we reduce from has distinct costs for any two neighboring solutions. The following lemma ensures that we can make this assumption.

▶ **Lemma 3.1.** *Distinct Max Cut-5/Flip is* PLS*-complete. More precisely, there exists a* PLS*-reduction from Max Cut-5/Flip to Distinct Max Cut-5/Flip.*

Unfortunately, this reduction is not tight. Hence, to prove the last two items of Theorem 1.3, simply applying the reductions from Figure 1 is not sufficient, as these properties (viz. PSPACE-completeness of the standard algorithm problem and the existence of certain hard instances) do not necessarily hold for Distinct Max Cut-5/Flip. We must instead separately prove that they hold for this problem. To accomplish this, we recall a construction by Monien and Tscheuschner [37] that shows these properties for Max Cut-4/Flip. It can be verified straightforwardly that the construction they use is already an instance of Distinct Max Cut-4/Flip.

In the remainder of this work, we present a sequence of tight reductions starting from Distinct Max Cut-5/Flip to all of our other considered problems. First, we reduce from Distinct Max Cut-5/Flip to Odd Half Pos NAE 3-SAT/Flip.

▶ **Lemma 3.2.** *There exists a tight* PLS*-reduction from Distinct Max Cut-5/Flip to Odd Half Pos NAE 3-SAT/Flip.*

As mentioned in Section 2.3, it may seem more straightforward to reduce from Min Bisection/Swap to Odd Min Bisection/Flip. The problem with this approach is that a solution to Odd Min Bisection/Flip can be locally optimal for two reasons: either no vertex can flip to obtain a cut of larger weight, or the vertices that could are in the smaller part of the partition. This makes the Flip neighborhood much less powerful than Swap in this problem variant; we were thus not able to find a direct reduction from Min Bisection/Swap. Instead, we apply a new technique that allows us to prove PLS-hardness for this very restricted problem.

We first prove PLS-hardness of Odd Half Pos NAE 3-SAT/Flip, and subsequently use existing reductions to obtain hardness of Odd Min Bisection/Flip. With the expressiveness of this SAT variant we gain a great deal of freedom to handle the problem restrictions. The main challenge is in encoding the restriction that the number of true and false variables must differ by exactly one without weakening the neighborhood.

Next, we briefly sketch and motivate some of the ideas in the reduction in more detail. See also Figure 2.

Sketch of Proof for Lemma 3.2. We first embed the Distinct Max Cut-5 instance, given by a weighted graph $G = (V, E)$, in Pos NAE SAT. This can be done rather straightforwardly, by a reduction used by Schäffer and Yannakakis [40]: each vertex becomes a variable, and an edge uv becomes a clause $\text{NAE}(u, v)$. This instance is directly equivalent to the original Distinct Max Cut-5 instance. We call these variables the *level 1 variables*, and the clauses the *level 1 clauses*. A level 1 clause $\text{NAE}(u, v)$ gets a weight $M \cdot w(uv)$ for some large integer M.

A solution to the original Distinct Max Cut-5 instance is obtained by placing the `true` level 1 variables in a feasible truth assignment on one side of the cut, and the `false` level 1 variables on the other side.

Given the reduction so far, suppose we have some locally optimal feasible truth assignment s. We partition the variables into the sets T and F of `true` and `false` variables; thus, (T, F) is the cut induced by s. Suppose $|T| = |F| + 1$. If no level 1 variable can flip from T to F, then also no vertex can flip from T to F in the induced cut. However, we run into trouble when there exists some $v \in F$ that can flip in the cut. Since $|F| < |T|$, we are not allowed to flip the level 1 variable v, and so the truth assignment may be locally optimal even though the induced cut is not.

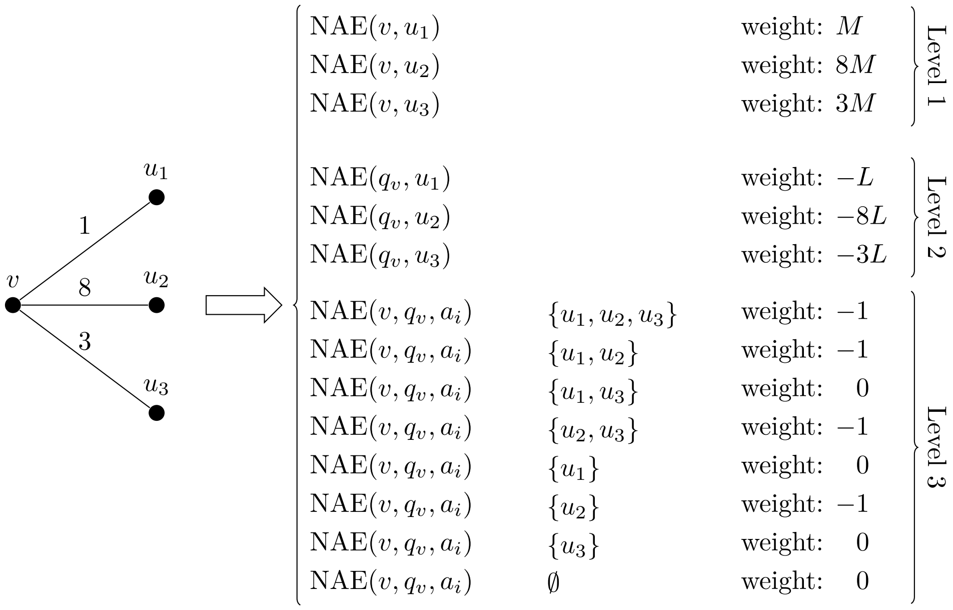

u_1
1
v
8
u_2
3
u_3
NAE(v, u_1) weight: M
NAE(v, u_2) weight: $8M$
NAE(v, u_3) weight: $3M$
Level 1
NAE(q_v, u_1) weight: $-L$
NAE(q_v, u_2) weight: $-8L$
NAE(q_v, u_3) weight: $-3L$
Level 2
NAE(v, q_v, a_i) $\{u_1, u_2, u_3\}$ weight: -1
NAE(v, q_v, a_i) $\{u_1, u_2\}$ weight: -1
NAE(v, q_v, a_i) $\{u_1, u_3\}$ weight: 0
NAE(v, q_v, a_i) $\{u_2, u_3\}$ weight: -1
NAE(v, q_v, a_i) $\{u_1\}$ weight: 0
NAE(v, q_v, a_i) $\{u_2\}$ weight: -1
NAE(v, q_v, a_i) $\{u_3\}$ weight: 0
NAE(v, q_v, a_i) $\emptyset$ weight: 0
Level 3

Figure 2 Schematic overview of the reduction used in the proof of Lemma 3.2. On the left we have a vertex $v \in V$ and its neighbors $\{u_1, u_2, u_3\}$ in a Max Cut instance, with weights on the edges between v and its neighbors. The NAE clauses on the right are the clauses constructed from v. In the actual reduction, these clauses are added for all level 3 variables. The right-most column shows the weights assigned to the clauses. The middle column shows for the level 3 clauses which subset of $N(v)$ corresponds to which clause. The constants L and M are chosen so that $1 \ll L \ll M$.

To deal with this situation, we will introduce two more levels of variables and clauses. The weights of the clauses at level i will be scaled so that they are much larger than those at level $i + 1$. In this way, changes at level i dominate changes at level $i + 1$, so that the Distinct Max Cut-5 instance can exist independently at level 1.

For each vertex $v \in V$, we add a variable q_v to the instance, and for each $u \in N(v)$, we add a clause NAE(q_v, u) with weight proportional to $-w(uv)$. We call these variables the *level 2 variables*, and these clauses the *level 2 clauses.*

Finally, we add $N = 2n + 1$ more variables $\{a_i\}_{i=1}^{N}$, which we call the *level 3 variables.* The number N is chosen so that for any truth assignment such that the number of `true` and `false` variables differ by one, there must exist a level 3 variable in the larger of the two sets. We then add more clauses as follows: for each level 3 variable a_i, for each $v \in V$, for each $Q \subseteq N(v)$, we add a clause $C_i(v, Q) = \text{NAE}(v, q_v, a_i)$. We give this clause a weight of -1 if and only if v can flip when each of the vertices in Q are present in the same half of the cut as v. We call these the *level 3 clauses*

Now consider the aforementioned situation, where a truth assignment s is locally optimal, but there exists some $v \in V \cap F$ that can flip in the induced cut. Carefully investigating the structure of such a locally optimal truth assignment shows that some level 2 or level 3 variable can flip for a strict improvement in the cost. This contradicts local optimality, and so we must conclude that locally optimal truth assignments induce locally optimal cuts, satisfying the essential property of PLS-reductions. ◀

As far as we are aware, this technique for overcoming size constraints in local search problems is novel. We believe that it may be useful to prove PLS-hardness results for simple heuristics for other size-constrained problems, such as balanced clustering problems.

A reduction from Pos NAE 3-SAT/Flip to Max Cut/Flip was provided by Schäffer and Yannakakis [40]. Since Max Cut is equivalent to Pos NAE 2-Sat, we can use the same reduction to reduce from Odd Half Pos NAE 3-SAT/Flip to Odd Half Pos NAE 2-SAT/Flip.

▶ **Lemma 3.3** (Schäffer and Yannakakis [40]). *There exists a tight* PLS*-reduction from Odd Half Pos NAE 3-SAT/Flip to Odd Half Pos NAE 2-SAT/Flip.*

While our reductions so far have used negative-weight clauses in Odd Half Pos NAE k-SAT/Flip, it may be of interest to have a PLS-completeness result also when all clauses have non-negative weight.

▶ **Corollary 3.4.** *Odd Half Pos NAE 2-SAT/Flip is* PLS*-complete even when all clauses have non-negative weight. More precisely, there exists a tight* PLS*-reduction from Odd Half Pos NAE 2-SAT/Flip to Odd Half Pos NAE 2-SAT/Flip where all clauses have non-negative weight.*

Finally, we reduce from Odd Half Pos NAE 2-SAT/Flip to Odd Min Bisection/Flip.

▶ **Lemma 3.5.** *There exists a tight* PLS*-reduction from Odd Half Pos NAE 2-SAT/Flip to both Odd Max Bisection/Flip and Odd Min Bisection/Flip.*

A reduction from Pos NAE 2-SAT/Flip to Max Cut/Flip is given by Schäffer and Yannakakis [40]. It is easy to see that this reduction also works with our constraint on the number of `true` and `false` variables, which yields a reduction to Odd Max Bisection/Flip.

4 Reduction to Clustering Problems

Armed with the PLS-completeness of Odd Min Bisection/Flip (see Lemma 3.5), we now proceed to prove hardness of the Euclidean clustering problems of interest.

k-Means. We provide a tight PLS-reduction from Odd Min Bisection/Flip to k-Means/Flip. This is done in three steps (see Figure 1). First, we show PLS-completeness of Densest Cut/Flip. The construction of the proof of our PLS-completeness of Densest Cut/Flip is rather simple (we only add a large set of isolated edges), but the analysis of the correctness is quite technical. Second, we show PLS-hardness of 2-Means/Flip by slightly adapting an NP-hardness reduction of 2-Means [3]. Finally, we extend this result to k-Means/Flip.

Now, we show PLS-completeness of Densest Cut/Flip. We impose the additional constraint that there are no isolated vertices in the reduced instance. This is a technical condition which is utilized in Lemma 4.3 for the PLS-hardness of k-Means/Flip.

For an illustration of the reduction of Lemma 4.1 we refer to Figure 3.

▶ **Lemma 4.1.** *There exists a tight* PLS*-reduction from Odd Max Bisection/Flip to Densest Cut/Flip without isolated vertices.*

Next, we show that also the closely related Sparsest Cut is PLS-complete under the Flip neighborhood. Note that Densest Cut and Sparsest Cut are both NP-hard [35]. Sparsest Cut is studied intensively in terms of approximation algorithms [5] and integrality gaps [25], and is used to reveal the hierarchical community structure of social networks [32] and in image segmentation [43].

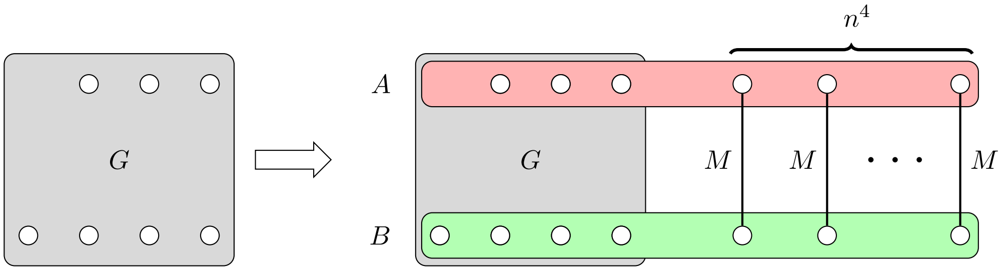


Figure 3 Schematic overview of the reduction used in the proof of Lemma 4.1. On the left side we have an instance of Odd Max Bisection/Flip and on the right side we have the corresponding instance of Densest Cut/Flip. The edges inside of G together with their weights are not depicted but are identical in both instances. Let (A, B) be the partition corresponding to some locally optimal solution of the Densest Cut/Flip instance. Then, A contains exactly one endpoint of each of the n^4 isolated edges and $\left|\, |A \cap V(G)| - |B \cap V(G)| \,\right| = 1$.

▶ **Corollary 4.2.** *There exists a tight* PLS*-reduction from Densest Cut/Flip to Sparsest Cut/Flip.*

The penultimate step is to show that 2-Means/Flip is PLS-hard. We achieve this by modifying a proof of NP-hardness of 2-Means by Alois et al. [3].

▶ **Lemma 4.3.** *There exists a tight* PLS*-reduction from Densest Cut/Flip without isolated vertices to 2-Means/Flip.*

Finally, we provide a generic reduction to show PLS-hardness for general k.

▶ **Lemma 4.4.** *For each $k \geq 2$, there exists a tight* PLS*-reduction from k-Means/Flip to $(k+1)$-Means/Flip.*

Now, Theorem 1.1 follows by applying the tight PLS-reductions according to Figure 1.

Squared Euclidean Max Cut. We construct a PLS-reduction from Odd Min Bisection/Flip to Squared Euclidean Max Cut/Flip. The reduction is largely based on the NP-hardness proof of Euclidean Max Cut of Ageev et al. [2]. The main difference is that we must incorporate the weights of the edges of the Odd Min Bisection/Flip instance into the reduction.

▶ **Lemma 4.5.** *There exists a tight* PLS*-reduction from Odd Min Bisection/Flip to Squared Euclidean Max Cut/Flip.*

With a few modifications, the proof can be adapted to a reduction to Euclidean Max Cut/Flip. The main challenge in adapting the proof is that the objective function is now of the form $\sum \|x - y\|$, rather than $\sum \|x - y\|^2$. However, by suitably modifying the coordinates of the points, the distances $\|x - y\|$ in the Euclidean Max Cut instance can take the same value as $\|x - y\|^2$ in the Squared Euclidean Max Cut instance.

▶ **Lemma 4.6.** *There exists a tight* PLS*-reduction from Odd Min Bisection/Flip to Euclidean Max Cut/Flip.*

5 Discussion

Theorems 1.1 and 1.3 show that no local improvement algorithm using the Flip heuristic can find locally optimal clusterings efficiently, even when $k = 2$. This result augments an earlier worst-case construction [33]. Theorem 1.2 demonstrates that finding local optima in Squared Euclidean Max Cut is no easier than for general Max Cut under the Flip neighborhood. Thus, the Euclidean structure of the problem yields no benefits with respect to the computational complexity of local optimization.

Smoothed Analysis. Other PLS-hard problems have yielded under smoothed analysis. Chiefly, Max Cut/Flip has polynomial smoothed complexity in complete graphs [4, 11] and quasi-polynomial smoothed complexity in general graphs [14, 18]. We hope that our results here serve to motivate research into the smoothed complexity of k-Means/Flip and Squared Euclidean Max Cut/Flip, with the goal of adding them to the list of hard local search problems that become easy under perturbations.

Reducing the Dimensionality. Our reductions yield instances of k-Means and (Squared) Euclidean Max Cut in $\Omega(n)$ dimensions. Seeing as our reductions cannot be obviously adapted for $d = o(n)$, we raise the question of whether the hardness of Squared Euclidean Max Cut/Flip and k-Means/Flip is preserved for $d = o(n)$. This seems unlikely for Squared Euclidean Max Cut/Flip for $d = O(1)$, since there exists an $O(n^{d+1})$-time exact algorithm due to Schulman [42]. A direct consequence of PLS-hardness for $d = f(n)$ would thus be an $O\big(n^{f(n)}\big)$-time general-purpose local optimization algorithm. Concretely, PLS-hardness for $d = \operatorname{polylog} n$ would yield a quasi-polynomial time algorithm for all problems in PLS.

For k-Means/Flip, the situation is similar: For $d = 1$, k-Means is polynomial-time solvable for any k. However, already for $d = 2$, the problem is NP-hard [31] when k is arbitrary. When both k and d are constants, the problem is again polynomial-time solvable, as an algorithm exists that finds an optimal clustering in time $n^{O(kd)}$ [21]. Thus, PLS-hardness for $kd \in O(f(n))$ would yield an $n^{O(f(n))}$-time algorithm for all PLS problems in this case.

Euclidean Local Search. There appear to be very few PLS-hardness results for Euclidean local optimization problems, barring the result of Brauer [13] and now Theorem 1.1 and Theorem 1.2. A major challenge in obtaining such results is that Euclidean space is very restrictive; edge weights cannot be independently set, so the intricacy often required for PLS-reductions is hard to achieve. Even in the present work, most of the work is done in a purely combinatorial setting. It is then useful to get rid of the Euclidean structure of the problem as quickly as possible, which we achieved by modifying the reductions of Ageev et al. [2] and Alois et al. [3].

With this insight, we pose the question of what other local search problems remain PLS-hard for Euclidean instances. Specifically, is TSP with the k-opt neighborhood still PLS-hard in Euclidean (or squared Euclidean) instances, for sufficiently large k? This is known to be the case for general metric instances for some large constant k [29] (recently improved to all $k \geq 17$ [22]), but Euclidean instances are still a good deal more restricted.

References

1 Emile Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003. doi:10.2307/j.ctv346t9c.

2 A. A. Ageev, A. V. Kel'manov, and A. V. Pyatkin. Complexity of the weighted max-cut in Euclidean space. *Journal of Applied and Industrial Mathematics*, 8(4):453–457, October 2014. doi:10.1134/S1990478914040012.

3 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, May 2009. doi:10.1007/s10994-009-5103-0.

4 Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 429–437, New York, NY, USA, June 2017. Association for Computing Machinery. doi:10.1145/3055399.3055402.

5 Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):5:1–5:37, April 2009. doi:10.1145/1502793.1502794.

6 David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed Analysis of the k-Means Method. *Journal of the ACM*, 58(5):19:1–19:31, October 2011. doi:10.1145/2027216.2027217.

7 David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SoCG '06, pages 144–153, New York, NY, USA, June 2006. Association for Computing Machinery. doi:10.1145/1137856.1137880.

8 David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, USA, January 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

9 Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design. *Operations Research*, 36(3):493–513, June 1988. doi:10.1287/opre.36.3.493.

10 P. Berkhin. A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 25–71. Springer, Berlin, Heidelberg, 2006. doi:10.1007/3-540-28349-8_2.

11 Ali Bibak, Charles Carlson, and Karthekeyan Chandrasekaran. Improving the Smoothed Complexity of FLIP for Max Cut Problems. *ACM Transactions on Algorithms*, 17(3):19:1–19:38, July 2021. doi:10.1145/3454125.

12 Y.Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1, July 2001. doi:10.1109/ICCV.2001.937505.

13 Sascha Brauer. Complexity of Single-Swap Heuristics for Metric Facility Location and Related Problems. In *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 116–127, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-57586-5_11.

14 Xi Chen, Chenghao Guo, Emmanouil V. Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local max-cut and binary max-CSP. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1052–1065, New York, NY, USA, June 2020. Association for Computing Machinery. doi:10.1145/3357713.3384325.

15 Robert Elsässer and Tobias Tscheuschner. Settling the Complexity of Local Max-Cut (Almost) Completely. In *International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 171–182, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-22006-7_15.

16 Michael Etscheid. *Beyond Worst-Case Analysis of Max-Cut and Local Search*. PhD thesis, Universitäts- und Landesbibliothek Bonn, August 2018. URL: https://bonndoc.ulb.uni-bonn.de/xmlui/handle/20.500.11811/7613.

17 Michael Etscheid and Heiko Röglin. Smoothed Analysis of the Squared Euclidean Maximum-Cut Problem. In *Algorithms - ESA 2015*, Lecture Notes in Computer Science, pages 509–520, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-662-48350-3_43.

18 Michael Etscheid and Heiko Röglin. Smoothed Analysis of Local Search for the Maximum-Cut Problem. *ACM Transactions on Algorithms*, 13(2):25:1–25:12, March 2017. doi:10.1145/3011870.

19 C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *19th Design Automation Conference*, pages 175–181, June 1982. doi:10.1109/DAC.1982.1585498.

20 J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. doi:10.2307/2346830.

21 Susumu Hasegawa, Hiroshi Imai, Mary Inaba, and Naoki Katoh. Efficient Algorithms for Variance-Based k-Clustering. *Proceedings of Pacific Graphics 1993*, February 2000.

22 Sophia Heimann, Hung P. Hoang, and Stefan Hougardy. The k-Opt algorithm for the Traveling Salesman Problem has exponential running time for $k \geq 5$. In *International Colloquium on Automata, Languages, and Programming*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84:1–84:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.84.

23 Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, June 2010. doi:10.1016/j.patrec.2009.09.011.

24 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, August 1988. doi:10.1016/0022-0000(88)90046-3.

25 Daniel M. Kane and Raghu Meka. A PRG for lipschitz functions of polynomials with applications to sparsest cut. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 1–10, New York, NY, USA, June 2013. Association for Computing Machinery. doi:10.1145/2488608.2488610.

26 T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002. doi:10.1109/TPAMI.2002.1017616.

27 B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970. doi:10.1002/j.1538-7305.1970.tb01770.x.

28 Christian Komusiewicz and Nils Morawietz. Finding 3-Swap-Optimal Independent Sets and Dominating Sets Is Hard. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.66.

29 M.W. Krentel. Structure in locally optimal solutions. In *30th Annual Symposium on Foundations of Computer Science*, pages 216–221, October 1989. doi:10.1109/SFCS.1989.63481.

30 S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982. doi:10.1109/TIT.1982.1056489.

31 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, July 2012. doi:10.1016/j.tcs.2010.05.034.

32 Charles F. Mann, David W. Matula, and Eli V. Olinick. The use of sparsest cuts to reveal the hierarchical community structure of social networks. *Social Networks*, 30(3):223–234, July 2008. doi:10.1016/j.socnet.2008.03.004.

33 Bodo Manthey and Jesse van Rhijn. Worst-Case and Smoothed Analysis of the Hartigan-Wong Method for k-Means Clustering. In *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.STACS.2024.52.

34 J. Matoušek. On Approximate Geometric k -Clustering. *Discrete & Computational Geometry*, 24(1):61–84, January 2000. doi:10.1007/s004540010019.

35 David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1):113–123, May 1990. doi:10.1016/0166-218X(90)90133-W.

36 Wil Michiels, Jan Korst, and Emile Aarts. *Theoretical Aspects of Local Search*. Monographs in Theoretical Computer Science, An EATCS Series. Springer, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-35854-1.

37 Burkhard Monien and Tobias Tscheuschner. On the Power of Nodes of Degree Four in the Local Max-Cut Problem. In *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 264–275, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-13073-1_24.

38 Christos H. Papadimitriou. The Complexity of the Lin–Kernighan Heuristic for the Traveling Salesman Problem. *SIAM Journal on Computing*, 21(3):450–465, June 1992. doi:10.1137/0221030.

39 Tim Roughgarden and Joshua R. Wang. The Complexity of the k-means Method. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.78.

40 Alejandro A. Schäffer and Mihalis Yannakakis. Simple Local Search Problems that are Hard to Solve. *SIAM Journal on Computing*, 20(1):56–87, February 1991. doi:10.1137/0220004.

41 I. J. Schoenberg. Metric spaces and positive definite functions. *Transactions of the American Mathematical Society*, 44(3):522–536, 1938. doi:10.1090/S0002-9947-1938-1501980-0.

42 Leonard J. Schulman. Clustering for edge-cost minimization (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 547–555, New York, NY, USA, May 2000. Association for Computing Machinery. doi:10.1145/335305.335373.

43 Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000. doi:10.1109/34.868688.

44 Matus Telgarsky and Andrea Vattani. Hartigan's Method: K-means Clustering without Voronoi. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 820–827. JMLR Workshop and Conference Proceedings, March 2010. URL: https://proceedings.mlr.press/v9/telgarsky10a.html.

45 Andrea Vattani. K-means Requires Exponentially Many Iterations Even in the Plane. *Discrete & Computational Geometry*, 45(4):596–616, June 2011. doi:10.1007/s00454-011-9340-1.

Online Multi-Level Aggregation with Delays and Stochastic Arrivals

Mathieu Mari
LIRMM, University of Montpellier, France

Michał Pawłowski
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland
Sapienza University of Rome, Italy

Runtian Ren
IDEAS NCBR, Warsaw, Poland
University of Wrocław, Poland

Piotr Sankowski
University of Warsaw, Poland
IDEAS NCBR, Poland
MIM Solutions, Warsaw, Poland

Abstract

This paper presents a new research direction for online Multi-Level Aggregation (MLA) with delays. Given an *edge-weighted rooted tree* T as input, a *sequence of requests* arriving at its vertices needs to be served in an online manner. A request r is characterized by two parameters: its *arrival time* $t(r) > 0$ and *location* $l(r)$ being a vertex in tree T. Once r arrives, we can either serve it immediately or postpone this action until any time $t > t(r)$. A request that has not been served at its arrival time is called pending up to the moment it gets served. We can serve several pending requests at the same time, paying a service cost equal to the weight of the subtree containing the locations of all the requests served and the root of T. Postponing the service of a request r to time $t > t(r)$ generates an additional delay cost of $t - t(r)$. The goal is to serve all requests in an online manner such that the total cost (i.e., the total sum of service and delay costs) is minimized. The MLA problem is a generalization of several well-studied problems, including the TCP Acknowledgment (trees of depth 1), Joint Replenishment (depth 2), and Multi-Level Message Aggregation (arbitrary depth). The current best algorithm achieves a competitive ratio of $O(d^2)$, where d denotes the depth of the tree.

Here, we consider a stochastic version of MLA where the requests follow a Poisson arrival process. We present a deterministic online algorithm that achieves a constant ratio of expectations, meaning that the ratio between the expected costs of the solution generated by our algorithm and the optimal offline solution is bounded by a constant. Our algorithm is obtained by carefully combining two strategies. In the first one, we plan periodic oblivious visits to the subset of frequent vertices, whereas, in the second one, we greedily serve the pending requests in the remaining vertices. This problem is complex enough to demonstrate a very rare phenomenon that "single-minded" or "sample-average" strategies are not enough in stochastic optimization.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, online network design, stochastic model, Poisson arrivals

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.49

Related Version *Full Version*: https://arxiv.org/abs/2404.09711 [57]

Funding This work is supported by ERC CoG grant TUgbOAT no 772346, NCN grant no 2020/37/B/ST6/04179, and NCN grant no 2022/45/B/ST6/00559.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 49; pp. 49:1–49:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Imagine the manager of a factory in charge of delivering products from the factory to the stores' locations. Once some products are in shortage for a store, its owner informs the factory for replenishment. From the factory's perspective, each time a service is created to deliver the products, a truck has to travel from the factory to the requested stores' locations and then return to the factory. A cost proportional to the total traveling distance is paid for this service. For the purpose of saving delivery costs, it is beneficial to accumulate the replenishment requests from many stores and then deliver the ordered products altogether in one service. However, this accumulated delay in delivering products may leave the stores unsatisfied, and the complaints will negatively influence future contracts between the stores and the factory. Typically, for each request ordered from a store, the time gap between ordering the products and receiving the products is known as the delay cost. The goal of the factory manager is to plan the delivery service schedule in an online manner such that the total service cost and the total delay cost are minimized.

The above is an example of an online problem called Multi-level Aggregation (MLA) with linear delays. Formally, the input is an edge-weighted rooted tree T and a sequence of requests, with each request r specified by an arrival time $t(r)$ and a location at a particular vertex. Once a request r arrives, its service does not have to be processed immediately but can be delayed to any time $t \geq t(r)$ at a delay cost of $t - t(r)$. The benefit of delaying requests is that several requests can be served together to save some service costs. To serve any set of requests R at time t, a subtree T' containing the tree root and locations of all the requests in R needs to be bought at a service cost equal to the total weight of edges in T'. The goal is to serve all requests in an online manner such that the total cost (i.e., the total service cost plus the total delay cost) is minimized.

Due to many real-life applications ranging from logistics, supply chain management, and data transmission in sensor networks, the MLA problem has recently drawn considerable attention [22, 16, 27, 13]. Besides, two classic problems in this area, TCP-acknowledgment (also known as a lot-sizing problem) and Joint Replenishment (JRP), are special cases of MLA with tree depths of 1 and 2, respectively. They were also extensively studied [32, 45, 63, 1, 47, 28, 21, 3, 60, 20]. Particularly for MLA, the current best online algorithm achieves a competitive ratio of $\mathrm{O}(d^2)$ [13], where d denotes the depth of the given tree.

However, it is often too pessimistic to assume no stochastic information on the input is available in practice – again, consider our delivery example. The factory knows all the historical orders and can estimate the request frequencies from the stores of all locations. It is reasonable to assume that the requests follow some stochastic distribution. Therefore, the following question is natural: *if the input follows some stochastic distribution, can we devise online algorithms for MLA with better performance guarantees?*

In this paper, we provide an affirmative answer to this question. We study a stochastic online version of MLA, assuming that the requests arrive following a Poisson arrival process. More precisely, the waiting time between any two consecutive requests arriving at the same vertex u follows an exponential distribution $\mathrm{Exp}(\boldsymbol{\lambda}(u))$ with parameter $\boldsymbol{\lambda}(u)$. In this model, the goal is to minimize the expected cost produced by an algorithm ALG for a random input sequence generated in a long time interval $[0, \tau]$. To evaluate the performance of ALG on stochastic inputs, we use the *ratio of expectations* (RoE) by comparing the expected cost of ALG with the expected cost of the optimal offline solution OPT (see Definition 6).

Our contribution. We prove that the performance guarantee obtained in the Poisson arrival model is significantly better compared with the current best competitiveness obtained in the adversarial model. More specifically, we propose a non-trivial deterministic online algorithm that achieves a constant ratio of expectations.

▶ **Theorem 1.** *For MLA with linear delays in the Poisson arrival model, there exists a deterministic online algorithm that achieves a constant ratio of expectations.*

Our algorithm is obtained by synergistically merging two carefully crafted strategies. The first strategy incorporates periodic oblivious visits to a subset of frequently accessed vertices, while the second strategy employs a proactive, greedy approach to handle pending requests in the remaining vertices. The complexity of this problem unveils a rare phenomenon – the inadequacy of "single-minded" or "sample-average" strategies in stochastic optimization. In this paper, we not only address this challenge but also point to further complex problems (such as facility location with delays [13] or online service with delays [9]) that require a similar approach in stochastic environments.

Previous works. The MLA problem has been only studied in the adversarial model. Bienkowski et al. [16] introduced a general version of MLA, assuming that the cost of delaying a request r by a duration t is $f_r(t)$. Here, $f_r(\cdot)$ denotes the delay cost function of r, which needs to be non-decreasing and satisfy $f_r(0) = 0$. They proposed an $\mathrm{O}(d^4 2^d)$-competitive online algorithm for this general delay cost version problem, where d denotes the tree depth [16, Theorem 4.2]. Later, the competitive ratio is further improved to $\mathrm{O}(d^2)$ by Azar and Touitou [13, Theorem IV.2] (for the general delay cost version). However, no matching lower bound has been found for the delay cost version of MLA – the current best lower bound on MLA (with delays) is 4 [16, Theorem 6.3], restricted to a path case with linear delays. Thus far, no previous work has studied MLA in the stochastic input model.

Organization. We give the notations and preliminaries in Section 2. As a warm-up, we study a special single-edge tree instance in Section 3. We show that there are two different situations, we call them *heavy* case and *light* case, and to achieve a constant ratio of expectations, the ideas for the two cases are different. In Section 4, we give an overview of our deterministic online algorithm (Theorem 1). This algorithm is the combination of two different strategies for two different types of instances.[1] In Section 5, we study the *heavy* instances as a generalization of heavy single-edge trees. In Section 6, we prove the main Theorem 1. In Section 7, we provide some extra related works in detail. We finish the paper by discussing some future directions in Section 8.

2 Notations and preliminaries

Weighted tree. Consider an edge-weighted tree T rooted at vertex $\gamma(T)$. We refer to its vertex set by $V(T)$ and its edge set by $E(T)$. When the context is clear, we denote the root vertex, vertex set, and edge set by γ, V, and E, respectively. We assume that each edge $e \in E$ has a positive weight w_e. For any vertex $u \in V$, except for the root vertex γ, we denote its *parent vertex* as $\mathrm{par}(u) \in V$, and $e_u = (u, \mathrm{par}(u))$ as the edge connecting u and its parent. We also define T_u as the subtree of T rooted at vertex u. In addition to the edge weights, we use the term *vertex weight* to refer to $w_u := w(e_u)$, where $u \in V$ and $u \neq \gamma$.

[1] Due to the space limits, all the results for the light instances are ignored here but can be found in the full version [57].

Given any two vertices $u, v \in V(T)$, we denote the path length from u to v in T by $d_T(u, v)$, i.e., it is the total weight of the edges along this path. Finally, we use $T[U]$ to denote the forest induced by vertices of $U \subseteq V(T)$ in T.

Problem description. An MLA *instance* is characterized by a tuple (T, σ), where T is a weighted tree rooted at γ and σ is a sequence of *requests.* Each request r is described by a tuple $(t(r), l(r))$ where $t(r) \in \mathbb{R}^+$ denotes r's *arrival time* and $l(r) \in V(T)$ denotes r's *location.* Thus, denoting by m the number of requests, we can rewrite $\sigma := (r_1, \dots, r_m)$ with the requests sorted in increasing order of their arrival times, i.e., $t(r_1) \leq t(r_2) \leq \dots \leq t(r_m)$. Given a sequence of requests σ, a *service* $s = (t(s), R(s))$ is characterized by the *service time* $t(s)$ and the set of requests $R(s) \subseteq \sigma$ it serves. A *schedule* S for σ is a sequence of services. We call schedule S *valid* for σ if each request $r \in \sigma$ is assigned a service $s \in S$ that does not precede r's arrival. In other words, a valid S for σ satisfies (i) $\forall\, s \in S\ \forall\, r \in R(s)$ $t(r) \leq t(s)$; (ii) $\{R(s) : s \in S\}$ forms a partition of σ. Given any MLA instance (T, σ), an MLA algorithm ALG needs to produce a valid schedule S to serve all the requests in σ. Particularly, for an online MLA algorithm ALG, at any time t, the decision to create a service to serve a set of pending request(s) cannot depend on the requests arriving after time t. For each request $r \in \sigma$, let $S(r)$ denote the service in S which serves r, i.e., for each $s \in S$, $S(r) = s$ if and only if $r \in R(s)$. Given a sequence of requests σ and a valid schedule S, the *delay* cost for a request $r \in \sigma$ is defined as $\texttt{delay}(r) := t(S(r)) - t(r)$. Using this notion, we define the *delay* cost for a service $s \in S$ and the *delay* cost for the schedule S as $\texttt{delay}(s) := \sum_{r \in R(s)} \texttt{delay}(r)$ and $\texttt{delay}(S) := \sum_{s \in S} \texttt{delay}(s)$. Besides, given any $r \in \sigma$, if it is pending at time t, let $\texttt{delay}(r, t) = t - t(r)$ denote its delay cost at this moment.

The *weight* (also called *service cost*) of a service $s \in S$, denoted by $\texttt{weight}(s, T)$, is defined as the weight of the minimal subtree of T that contains root γ and all locations of requests $R(s)$ served by s. The *weight* (or *service cost*) of a schedule S is defined as $\texttt{weight}(S, T) := \sum_{s \in S} \texttt{weight}(s, T)$. To compute the *cost* of a service s, we sum its delay cost and weight, i.e., $\texttt{cost}(s, T) := \texttt{delay}(s) + \texttt{weight}(s, T)$. Similarly, we define the *cost* (or *total cost*) of a schedule S for σ as $\texttt{cost}(S, T) := \texttt{delay}(S) + \texttt{weight}(S, T)$. When the context is clear, we simply write $\texttt{cost}(S) = \texttt{cost}(S, T)$. Moreover, given an MLA instance (T, σ), let ALG(σ) denote the schedule of algorithm ALG for σ and let OPT(σ) denote the optimal schedule for σ with minimum total cost. Note that without loss of generality, we can assume that no request in σ arrives at the tree root γ since such a request can be served immediately at its arrival with zero cost.

Poisson arrival model. Instead of using an adversarial model, we assume that the requests arrive according to some stochastic process. A *stochastic instance* is characterized by a tuple $(T, \boldsymbol{\lambda})$, where T denotes an edge-weighted rooted tree, and $\boldsymbol{\lambda} : V(T) \to \mathbb{R}_+$ is a function that assigns each vertex $u \in V(T)$ an *arrival rate* $\boldsymbol{\lambda}(u) \geq 0$. With no loss we assume $\boldsymbol{\lambda}(\gamma(T)) = 0$, i.e., no request arrives at the tree root. Formally, such a tuple defines the following process.

▶ **Definition 2** (Poisson arrival model). *Given any stochastic MLA instance* $(T, \boldsymbol{\lambda})$ *and any* $\tau > 0$*, we say that a (random)* requests sequence σ *follows a* Poisson arrival model *over time interval* $[0, \tau]$*, if (i) for each vertex* $u \in V(T)$ *with* $\boldsymbol{\lambda}(u) > 0$ *the* waiting time *between any two consecutive requests arriving at* u *follows an* exponential distribution *with parameter* $\boldsymbol{\lambda}(u)$*;* [2] *(ii) variables representing waiting times are mutually independent; (iii) all the requests in* σ *arrive within time interval* $[0, \tau]$*. We denote this fact by writing* $\sigma \sim (T, \boldsymbol{\lambda})^\tau$*.*

[2] For the first request r arriving at u, the waiting time from 0 to $t(r)$ follows the distribution $\text{Exp}(\boldsymbol{\lambda}(u))$. Similarly, for the last request r' arriving at u, denoting by $W_{r'} \sim \text{Exp}(\boldsymbol{\lambda}(u))$ its waiting time, we require that $\tau - t(r') < W_{r'}$.

Given any subtree T' of T, we use both $\boldsymbol{\lambda}|_{T'}$ and $\boldsymbol{\lambda}|_{V(T')}$ to denote the arrival rates restricted to the vertices of T'. Similarly, given a random sequence of requests $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, we use $\sigma|_{T'} \subseteq \sigma$ and $\sigma|_I \subseteq \sigma$ for $I \subseteq [0, \tau]$ to denote the sequences of all requests in σ that arrive inside the subtree T' and within the time interval I, respectively.

In the following, we introduce three more properties of the Poisson arrival model. To simplify their statements, we denote the random variable representing the number of requests in sequence $\sigma \sim (T, \boldsymbol{\lambda})^\tau$ by $N(\sigma)$. The first property describes the expected value of $N(\sigma)$ for a fixed time horizon τ. The second one describes our model's behavior under the assumption that we are given the value of $N(\sigma)$. Finally, the third one presents the value of the expected waiting time generated by all the requests arriving before a fixed time horizon. All the proofs can be found in [62] or the full version of this paper [57].

▶ **Proposition 3.** *Given any stochastic MLA instance $(T, \boldsymbol{\lambda})$ and a random sequence of requests $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, it holds that: (i) $N(\sigma) \sim Poiss(\boldsymbol{\lambda}(T) \cdot \tau)$; (ii) $\mathbb{E}[N(\sigma) \mid \sigma \sim (T, \boldsymbol{\lambda})^\tau] = \boldsymbol{\lambda}(T) \cdot \tau$; (iii) if $\boldsymbol{\lambda}(T) \cdot \tau \geq 1$, then $\mathbb{P}(N(\sigma) \geq \mathbb{E}[N(\sigma)]) \geq 1/2$.*

▶ **Proposition 4.** *Given n requests arriving during time interval $[0, \tau]$ according to Poisson arrival model, the n arrival times (in sequence) have the same distribution as the order statistics corresponding to n independent random variables uniformly distributed over $[0, \tau]$.*

▶ **Proposition 5.** *Given any stochastic MLA instance $(T, \boldsymbol{\lambda})$ and a random sequence of requests $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, the expected delay cost of all the requests in $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, assuming that no service was issued before τ, is $\mathbb{E}[\sum_{i=1}^{N(\sigma)} (\tau - t(r_i))] = \mathbb{E}[N(\sigma)] \cdot \tau/2 = \boldsymbol{\lambda}(T) \cdot \tau^2/2$.*

Benchmark description. To measure the performance of an online algorithm ALG in this stochastic version of MLA, we use the *ratio of expectations.* Let $\mathbb{E}[\texttt{cost}(\text{ALG}(\sigma), T)]$ denote the expected cost of the schedule ALG generates for a random sequence $\sigma \sim (T, \boldsymbol{\lambda})^\tau$.

▶ **Definition 6** (ratio of expectations)**.** *An online algorithm* ALG *has a ratio of expectations (RoE) $C \geq 1$ if $\overline{\lim_{\tau \to \infty}} \frac{\mathbb{E}[cost(ALG(\sigma),T)]}{\mathbb{E}[cost(\text{OPT}(\sigma),T)]} \leq C$ for any stochastic MLA instance $(T, \boldsymbol{\lambda})$.*

3 Warm-up: single edge instances

We start by considering the case of a single-edge tree in the stochastic model. That is, we fix a tree T that consists of a single edge $e = (u, \gamma)$ of weight $w > 0$ and denote the arrival rate of u by $\lambda > 0$. In such a setting, the problem of finding the optimal schedule to serve the requests arriving at vertex u is known as *TCP acknowledgment.* It is worth mentioning that in the adversarial setting, a 2-competitive deterministic and a $(1 - 1/e)^{-1}$-competitive randomized algorithms are known for this problem [32, 45].

Let us stress that the goal of this section is not to improve the best-known competitive ratio for a single-edge case but to illustrate the efficiency of two opposite strategies and introduce some important concepts of this paper. The first strategy, called the *instant strategy*, is to serve each request as soon as it arrives. Intuitively, this approach is efficient when the requests are not so frequent so that, on average, the cost of delaying a request to the arrival time of the next request is enough to compensate for the service cost. The second strategy, called the *periodic approach*, is meant to work in the opposite case where requests are frequent enough so that it is worth grouping several of them for the same service. In this way, the weight cost of a service can be shared between the requests served. Assuming that requests follow some stochastic assumptions, it makes sense to enforce that services are ordered at regular time steps, where the time between any two consecutive services is a fixed number p, which depends only on the instance's parameters.

There are two challenges here. First, when should we use each strategy? Second, what should be the value of p that optimizes the performance of the periodic strategy? For the first one, we show that it depends on the value of $\pi := w\lambda$ that we call the *heaviness* of the instance. Specifically, we show that if $\pi > 1$, i.e., the instance is *heavy*, and the periodic strategy is more efficient. On the other hand, if $\pi \leq 1$, the instance is *light*, and the instant strategy is essentially better. For the second one, we show that the right value for the period, up to a constant in the ratio of expectations, is $p = \sqrt{2w/\lambda}$. With no loss, in what follows, we assume that the time horizon τ is always a multiple of the period chosen, which simplifies the calculation and does not affect the ratio of expectations.

▶ **Lemma 7.** *Given a stochastic instance where the tree is a single edge of weight w and the leaf has an arrival rate $\lambda > 0$, let $\pi = w\lambda$ and let σ be a random sequence of requests of duration $\tau > 0$. Then,*

1. *the instant strategy on σ has an expected cost of $\tau\pi$;*
2. *the periodic strategy on σ, with period $p = \sqrt{2w/\lambda}$, has an expected cost of $\tau\sqrt{2\pi}$.*

Note that the instant strategy incurs an expected cost equal to the expected number of requests arriving within the time horizon τ multiplied by the cost of serving one. By Proposition 3, we have that on average $\lambda\tau$ requests arrive within the time interval $[0, \tau]$. Thus, since the cost of serving one equals w, the total expected cost is $\lambda\tau w = \tau\pi$. For the periodic strategy, we know that within each period $p = \sqrt{2w/\lambda}$, we generate the expected delay cost of $\frac{1}{2} \cdot \lambda p^2 = w$ (Proposition 5). The service cost we pay at the end of each period equals w as well. Thus, the total expected cost within $[0, \tau]$ is equal to $\frac{\tau}{p} \cdot 2w = \tau \cdot \sqrt{2\lambda w} = \tau\sqrt{2\pi}$, which ends the proof.

We now compare these expected costs with the expected cost of the optimal offline schedule. The bounds obtained imply that the instant strategy has constant RoE when $\pi \leq 1$, and the periodic strategy (with $p = \sqrt{2w/\lambda}$) has a constant RoE when $\pi > 1$.

▶ **Lemma 8.** *Given a stochastic instance where the tree is a single edge of weight w and the leaf has an arrival rate $\lambda > 0$, let $\pi = w\lambda$ and let σ be a random sequence of requests of duration $\tau > 0$. For the lower bounds on the optimal offline schedule,* OPT(σ)*, it holds that*

1. *if $\pi \leq 1$, it has an expected cost of at least $\frac{1-e^{-1}}{2}\tau\pi$;*
2. *if $\pi > 1$, it has an expected cost of at least $\frac{3}{16}\tau\sqrt{2\pi}$.*

Due to the space limit, we only provide a proof sketch of Lemma 8 as follows. The main idea is to partition the initial time horizon $[0, \tau]$ into a collection of shorter intervals $\{I_1, I_2, \ldots, I_k\}$ of length p each, for some value p that will be defined later. Denoting by $\sigma_i := \sigma|_{I_i}$ for $i \in [k]$, we know that all σ_i are independent and follow the same Poisson arrival model $(T, \boldsymbol{\lambda})^p$. Let $D(\sigma_1)$ denote the total delay cost of σ_1 at time p when no services are issued during $[0, p]$. Note that OPT either serves some requests during $[0, p]$ and incurs the service cost of at least w or issues no services during $[0, p]$ and pays the delay cost of $D(\sigma_1)$. The total cost of OPT within $[0, p]$ is thus at least $\min(w, D(\sigma_1))$ and hence

$$\mathbb{E}\big[\texttt{cost}(\text{OPT}(\sigma))\big] \geq \tfrac{\tau}{p} \cdot \mathbb{E}\big[\min(w, D(\sigma_1)) \mid \sigma_1 \sim (T, \boldsymbol{\lambda})^p\big].$$

Define $U_j = p - t(r_j)$ for the j-th request r_j in σ_1. If $\pi \leq 1$, we choose $p = \frac{1}{\lambda}$, for which $\mathbb{P}(N(\sigma_1) \geq 1) = 1 - e^{-1}$ and $\mathbb{E}[\min(w, U_1)] \geq \frac{w}{2}$. This implies that

$$\mathbb{E}[\texttt{cost}(\text{OPT}(\sigma))] \geq \tfrac{1-e^{-1}}{2} \cdot \tau \cdot \lambda w.$$

If $w\lambda > 1$, we set $p = \sqrt{2w/\lambda}$ and $n_0 = \lceil \lambda p \rceil$, for which $\mathbb{P}(N(\sigma) \geq n_0) \geq \frac{1}{2}$ and

$$\mathbb{E}[\min(w, \sum_{j=1}^{n_0} U_j)] \geq 1 - \frac{w}{2n_0 p} \geq \tfrac{3}{4}.$$

This implies

$$\mathbb{E}[\texttt{cost}(\mathrm{OPT}(\sigma))] \geq \tfrac{3}{16} \cdot \tau \cdot \sqrt{2w\lambda}.$$

See the appendix in the full version [57] for a detailed proof.

4 Overview

We now give an overview of the following sections. Inspired by the two strategies for the single edge instance, we define two types of stochastic instances: the *light* instances, for which the strategy of serving requests instantly achieves a constant RoE, and the *heavy* instances, for which the strategy of serving requests periodically achieves a constant RoE. Heavy and light instances are defined precisely below (Definitions 9 and 13) and generalize the notions of heavy and light single-edge trees studied in the previous section.

We define the light instances by extending the notion of *heaviness* for an arbitrary tree.

▶ **Definition 9.** *A stochastic MLA instance $(T, \boldsymbol{\lambda})$ is called light if $\pi(T, \boldsymbol{\lambda}) \leq 1$, where $\pi(T, \boldsymbol{\lambda}) := \sum_{u \in V(T)} \boldsymbol{\lambda}(u) \cdot d(u, \gamma(T))$ is called the* heaviness *of the instance.*

For a light instance, serving the requests immediately at the their arrival time achieves a constant ratio of expectations. We refer to the schedule produced with this strategy (see Algorithm INSTANT in the full version [57]) on a sequence of requests σ by INSTANT(σ).

▶ **Theorem 10.** *INSTANT has O(1)-RoE for light instances.*

The theorem follows immediately from the following two lemmas (due to space limit, see the full version [57] for their proofs).

▶ **Lemma 11.** *If $(T, \boldsymbol{\lambda})$ is light, then $\mathbb{E}\left[\mathit{cost}(\mathrm{INSTANT}(\sigma)) \mid \sigma \sim (T, \boldsymbol{\lambda})^{\tau}\right] = \tau \cdot \pi(T, \boldsymbol{\lambda})$.*

▶ **Lemma 12.** *If $(T, \boldsymbol{\lambda})$ is light, then $\mathbb{E}\left[\mathit{cost}(\mathrm{OPT}(\sigma)) \mid \sigma \sim (T, \boldsymbol{\lambda})^{\tau}\right] = \Omega(1) \cdot \tau \cdot \pi(T, \boldsymbol{\lambda})$.*

We now turn our attention to heavy instances. An instance $(T, \boldsymbol{\lambda})$ is *heavy* if for every subtree $T' \subseteq T$, we have $\pi(T', \boldsymbol{\lambda}) > 1$. By monotonicity of $\pi(\cdot, \boldsymbol{\lambda})$, we obtain the following equivalent definition. Recall that for a vertex $u \in V(U)$, w_u denotes the weight of the edge incident to u on the path from $\gamma(T)$ to u.

▶ **Definition 13.** *A stochastic MLA instance $(T, \boldsymbol{\lambda})$ is called* heavy *if $w_u \geq 1/\boldsymbol{\lambda}(u)$ for all $u \in V(T)$ with $\boldsymbol{\lambda}(u) > 0$.*

To give some intuition, suppose that u is a vertex of a heavy instance, and r and r' are two consecutive (random) requests located on u. Then, the expected duration between their arrival times is $1/\boldsymbol{\lambda}(u) < w_u$. This suggests that to minimize the cost, we should, on average, gather r and r' into the same service in order to avoid paying twice the weight cost w_u. Since we expect services to serve a group of two or more requests, our stochastic assumptions suggest that the services must follow some form of regularity.

In Section 5, we present an algorithm called PLAN, that given a heavy instance $(T, \boldsymbol{\lambda})$, computes for each vertex $u \in V(T)$ a period $p_u > 0$, and will serve u at every time that is a multiple of p_u. One intuitive property of these periods $\{p_u : u \in V(T)\}$ is that the longer the distance to the root, the longer the period. While losing only a constant fraction of the expected cost, we choose the periods to be (scaled) powers of 2. This enables us to optimize the weights of the services in the long run. One interesting feature of our algorithm is that it acts "blindly": the algorithm does not need to know the requests, but only the arrival rate of each point. Indeed, our algorithm may serve a vertex where there are no pending requests. For the details of the PLAN algorithm, see Section 5.

▶ **Theorem 14.** *PLAN has O(1)-RoE for heavy instances.*

We remark that light instances and heavy instances are not complementary: there are instances that are neither light nor heavy.[3] In Section 6, we focus on the general case of arbitrary instances. The strategy here is to partition the tree (and the sequence of requests) into two groups of vertices (two groups of requests) so that the first group corresponds to a light instance where we can apply the instant strategy while the second group corresponds to a heavy instance where we can apply a periodic strategy. However, this correspondence for the heavy group is not straightforward. For this, we need to define an *augmented tree* that is a copy of the original tree, with the addition of some carefully chosen vertices. Each new vertex is associated with a subset of vertices of the original tree called *part*. We then define an arrival rate for each of these new vertices that is equal to the sum of the arrival rates of the vertices in the corresponding part. We show that this defines a heavy instance on which we can apply the algorithm PLAN. For each service made by PLAN on each of these new vertices, we serve all the pending requests in the corresponding part. The full description of this algorithm, called GEN, is given in Section 6. We show that this algorithm achieves a constant ratio of expectations.

▶ **Theorem 15.** *GEN has O(1)-RoE for arbitrary stochastic instances.*

5 Heavy instances

In this section, we analyze heavy MLA instances. Recall that an instance $(T, \boldsymbol{\lambda})$ is called heavy if $w_u \geq 1/\boldsymbol{\lambda}(u)$ for all $u \in V(T)$ with $\boldsymbol{\lambda}(u) > 0$. To serve this type of MLA instances, we devise an algorithm PLAN and prove that it achieves a constant ratio of expectations. Our approach can be seen as a generalization of the periodical strategy for a single-edge case. Once again, we serve the requests periodically, although this time, we may assign different periods for different vertices. Intuitively, vertices closer to the root and having a greater arrival rate should be served more frequently. For this reason, PLAN generates a partition P of a given tree T into a family of subtrees (clusters) and assigns them specific periods.

The partition procedure allows us to analyze each cluster separately. Thus, it is sufficient to estimate the performance of PLAN algorithm when restricted to a given subtree $T' \in P$. To lower bound the cost generated by OPT on T', we split the weight of T' among its vertices using a saturation procedure. Then we say that for each vertex v, the optimal algorithm either covers the delay cost of all the requests arriving at v within a given time horizon or it pays some share of the service cost. The last step is to round the periods assigned to the subtrees in P to minimize the cost of PLAN. In what follows, we present the details.

Periodical algorithm PLAN

As mentioned before, the main idea is to split tree T rooted at vertex γ into a family of subtrees and serve each of them periodically. In other words, we aim to find a partition $P = \{T_1, T_2, \ldots, T_k\}$ of T where each subtree T_i besides the one containing γ is rooted at the leaf vertex of another subtree. At the same time, we assign each subtree T_i some period p_i. To decide how to choose the values of p_is, recall how we picked the period for a single-edge case. In that setting, for the period p, we had an equality between the expected delay cost $\lambda/2 \cdot p^2$ at the leaf u and the weight w of the edge. Thus, the intuition behind the PLAN algorithm is as follows.

[3] There exists a stochastic MLA instance where the ratio of expectations are both unbounded if INSTANT or PLAN is directed applied to deal with. See the appendix in the full version [57] for details.

We start by assigning each vertex $v \in T$ a process that saturates the edge connecting it to the parent at the pace of $\boldsymbol{\lambda}(v)/2 \cdot t^2$, i.e., within the time interval $[t, t+\epsilon]$ it saturates the weight of $\boldsymbol{\lambda}(v)/2 \cdot ((t+\epsilon)^2 - t^2)$. By saturation, here we mean assigning a part of the edge weight to the vertex. In other words, at time t each vertex v has a budget that is equal to its expected delay cost, i.e., $\boldsymbol{\lambda}(v)/2 \cdot t^2$, and uses it to cover some part of the edge weight. Whenever an edge gets saturated, the processes that contributed to this outcome start working together with the processes that are still saturating the closest ancestor edge. As the saturation procedure within the whole tree T reaches the root γ, we cluster all the vertices corresponding to the processes that made it possible into the first subtree T_1. Moreover, we set the period of T_1 to the time it got saturated. After this action, we are left with a partially saturated forest having the leaves of T_1 as the root vertices. The procedure, however, follows the same rules, splitting the forest further into subtrees $T_2, \ldots, T_k$.

To simplify the formal description of our algorithm, we first introduce some new notations. Let $p(v)$ denote the saturation process defined for a given vertex v. As mentioned before, we define it to saturate the parent edge at the pace of $\boldsymbol{\lambda}(v)/2 \cdot t^2$. Moreover, we extend this notation to the subsets of vertices, i.e., we say that $p(S)$ is the saturation process where all the vertices in S cooperate to cover the cost of an edge. The pace this time is equal to $\boldsymbol{\lambda}(S)/2 \cdot t^2$. To trace which vertices cooperate at a given moment and which edge they saturate, we denote the subset of vertices that v works with by $S(v)$ and the edge they saturate by $e(v)$. We also define a method $\texttt{join}(u, v)$ that takes as the arguments two vertices and joins the subsets they belong to. It can be called only when the saturation process of $S(u)$ reaches v. Formally, at this moment, the `join` method merges subset $S(u)$ with $S(v)$ and sets $e(v)$ as the outcome of the function e on all the vertices in the new set. It also updates the saturation pace of the new set. We present the pseudo-code for PLAN as Algorithm 1 and an example as a visual support shown in Figure 1, followed by some properties of the partition generated by this algorithm (see the appendix in the full version [57] for the detailed proof) and the lower bounding scheme (Lemma 17).

▶ **Proposition 16.** *Let $(T, \boldsymbol{\lambda})$ be a heavy instance and let $P = \{T_1, T_2, \ldots, T_k\}$ be the partition generated on it by Algorithm 1. We denote the period corresponding to T_i by p_i. Assuming that T_is are listed in the order they were added to P, it holds that:*

1. *each T_i is a rooted subtree of T;*
2. *the periods are increasing, i.e., $1 \leq p_1 \leq p_2 \leq \ldots \leq p_k$;*
3. *each vertex $v \in T_i$ saturated exactly $\boldsymbol{\lambda}(v)/2 \cdot p_i^2$ along the path to the root of T_i.*

▶ **Lemma 17.** *Let $(T, \boldsymbol{\lambda})$ be a heavy instance. We denote the partition generated for it by Algorithm 1 by $P = \{T_1, T_2, \ldots, T_k\}$ and the period corresponding to T_i by p_i for all $i \in [k]$. Let T_i be any subtree in P, and let us define σ_i as a random sequence of requests arriving within the MLA instance restricted to T_i over a time horizon τ. We assume that τ is a multiple of p_i. It holds that $\mathbb{E}[\texttt{cost}(\mathrm{OPT}(\sigma_i), T_i) \mid \sigma_i \sim (T_i, \boldsymbol{\lambda}|_{T_i})^\tau] \geq \frac{3}{16} \cdot w(T_i) \cdot \frac{\tau}{p_i}$.*

The main idea is to use the same approach as in Section 3 and lower bound the cost incurred by OPT within a shorter time interval. The proof of Lemma 17 can be found in the full version [57].

PLAN has RoE $\leq 64/3 = 21.34$ for heavy instances

In Algorithm 1, each subtree T_i is served periodically with periods p_i. In this setting, to serve any cluster besides the one containing the root vertex γ, not only do we need to cover the service cost of the cluster vertices but also the cost of the path connecting them to γ. Since

Algorithm 1 PLAN (part I).

```
Input: an heavy instance $(T, \boldsymbol{\lambda})$ with tree $T$ rooted at $\gamma$
Output: a partition $P = \{T_1, T_2, \ldots, T_k\}$ of $T$, each subtree $T_i$ assigned a period $p_i$
let $R$ be the set of roots, initially $R = \{\gamma\}$
for each vertex $v \in V(T)$ do
    define the saturation process $p(v)$ as described before
    set $S(v) := \{v\}$ and $e(v) := \mathrm{par}(v)$
end
start the clock at time 0
while there exist some unclustered vertices in $T$ do
    wait until the first time $t_e$ when an edge $e = (u, v)$ gets saturated
    if $v \notin R$ then
        join$(u, v)$
    end
    else
        add cluster $C := S(u) \cup \{v\}$ to partition $P$
        set the period $p$ for $C$ to be equal to $t_e$
        set the saturation pace for $C$ to 0
        extend $R$ by the leaves in $C$
    end
end
```

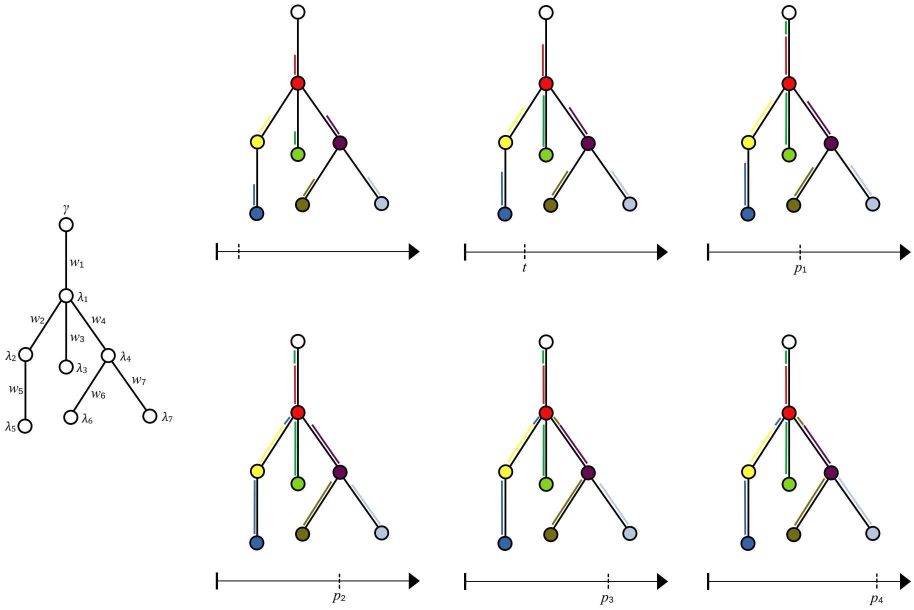


Figure 1 Here is an example to show how Algorithm 1 works on an heavy instance. Given the tree consisting of 7 vertices (with $w_i \geq 1/\lambda_i$ for each vertex $i \in [7]$ marked in different color), we use the length of the colored line to denote the saturated amount (i.e., $\boldsymbol{\lambda}_i/2 \cdot t^2$) of a vertex i at any time t. At time p_1, the subtree T_1 including vertices 1 and 3 is determined; similarly, T_2 includes vertices 2 and 5 at time t_2; T_3 includes vertices 4 and 6 at time p_3; and T_4 includes vertex 7 at time p_4.

we only know how to lower bound the cost incurred by OPT on the clusters, we improve the PLAN algorithm to get rid of this issue. The idea is to round the periods p_is to be of form $2^{e_i}p_1$ for some positive integers e_i. Thus, whenever we need to serve some cluster S_i, we know that we get to serve all the clusters generated before it as well. Due to the space limits, the formal pseudo-codes of the rounding procedures (Algorithm PLAN, part 2) and serving a random sequence of requests (Algorithm PLAN, part 3) can be found in the full version [57].

Let $(T, \boldsymbol{\lambda})$ be a heavy instance and let $P = \{T_1, \ldots, T_k\}$ be the partition generated for it by Algorithm 1. Let $(p_1, \ldots, p_k)$ and $(\hat{p}_1, \ldots, \hat{p}_k)$ denote the periods obtained from Algorithm 1 and rounded periods (by Algorithm PLAN, part 2), respectively. Now we analyze the cost of PLAN on $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, where the time horizon τ is a multiple of $2\hat{p}_k$. Since we align the periods to be of form $2^l\hat{p}_1$ for some positive integer l, whenever PLAN serves some tree T_i, it serves all the trees containing the path from T_i to γ at the same time. Thus, the service cost can be estimated on the subtree level. Moreover, since for each $i \in [k]$ we round p_i such that $\hat{p}_i \leq p_i$, the expected delay cost incurred within $[0, \hat{p}_i]$ does not exceed $w(T_i)$. Denoting by $\sigma_i \sim (T_i, \boldsymbol{\lambda}|_{T_i})^{\hat{p}_i}$ for $i \in [k]$, we have

$$\mathbb{E}[\texttt{cost}(\text{PLAN}(\sigma), T)] = \textstyle\sum_{i=1}^{k} \frac{\tau}{\hat{p}_i} \cdot \mathbb{E}[\texttt{cost}(\text{PLAN}(\sigma_i), T_i)] = \sum_{i=1}^{k} \frac{\tau}{\hat{p}_i} \cdot 2w(T_i).$$

On the other hand, the expected cost of OPT for σ, i.e., $\mathbb{E}[\texttt{cost}(\text{OPT}(\sigma), T)]$, is at least

$$\textstyle\sum_{i=1}^{k} \mathbb{E}[\texttt{cost}(\text{OPT}(\sigma|_{T_i}), T_i)] \geq \sum_{i=1}^{k} \frac{\tau}{p_i} \frac{3}{16} w(T_i).$$

By definition, it holds that $p_i < 2\hat{p}_i$ for $i \in [k]$. We can rewrite the above as

$$\mathbb{E}[\texttt{cost}(\text{OPT}(\sigma), T)] > \textstyle\sum_{i=1}^{k} \frac{\tau}{2\hat{p}_i} \frac{3}{16} w(T_i) = \frac{3}{32} \sum_{i=1}^{k} \frac{\tau}{\hat{p}_i} w(T_i),$$

and hence establishing RoE(PLAN) = 64/3.

6 General instances

Now we devise an algorithm GEN for an arbitrary stochastic instance $(T, \boldsymbol{\lambda})$, which achieves a constant ratio of expectations. The main idea is to distinguish two types of requests and apply a different strategy for each type. The first type is the requests that are located close to the root. These requests will be served immediately at their arrival times, i.e., we apply INSTANT to the corresponding sub-sequence. The second type includes all remaining requests, and they are served in a periodic manner. To determine the period of these vertices, we will use the algorithm PLAN on a specific heavy instance $(T', \boldsymbol{\lambda}^h)$. The construction of this heavy instance relies on a partition of the vertices of T into *balanced parts*. Intuitively, a part is balanced when it is light (or close to being light), but if we merge all vertices of the part into a single vertex whose weight corresponds to the average distance to the root of the part, then we obtain a heavy edge. This "merging" process is captured by the construction of the *augmented tree* T', which is part of the heavy instance. The augmented tree is essentially a copy of T with the addition of one (or two) new vertices for each balanced part.

Once determining the corresponding heavy instance, we can compute the periods of each vertex of the heavy instance using PLAN. The vertex period in the original instance is equal to the corresponding vertex period in the heavy instance. For the full description of GEN, see Algorithm 2 in the appendix of the full version [57]. The main challenge is to analyze the ratio of expectations and in particular, to establish good lower bounds on the expected cost of the optimal offline schedule. Due to the space limits, check the full version [57], where we prove two lower bounds that depend on the heaviness of each part of the balanced partition.

Notations and additional assumptions. Given the edge-weighted tree T rooted at γ and a set of vertices $U \subseteq V(T)$, $T[U]$ denotes the forest induced on U in T. We say that a subset $U \subseteq V(T)$ is *connected* if $T[U]$ is connected (i.e., $T[U]$ is a subtree of T but not a forest). If $U \subseteq V(T)$ is connected, we write $\gamma(U) = \gamma(T[U])$ to denote the root vertex of $T[U]$, i.e., the vertex in U which has the shortest path length to γ in the original tree T. Given any vertex $u \in V(T)$, let $V_u \subseteq V(T)$ denote all the descendant vertices of u in T (including u). For simplicity, set $w_{\gamma(T)} = \infty$. Given $T = (V, E)$, $\boldsymbol{\lambda} : V(T) \to \mathbb{R}_+$ and $U \subseteq V$, we denote $\boldsymbol{\lambda}|_U : U \to \mathbb{R}_+$ such that $\boldsymbol{\lambda}|_U(u) = \boldsymbol{\lambda}(u)$ for each $u \in U$. For a sequence of requests $\sigma \sim (T, \boldsymbol{\lambda})$, we use $\sigma|_U = \{r \in \sigma \mid \ell(r) \in U\}$ to denote the corresponding sequence for $T[U]$. In this section, we assume $\boldsymbol{\lambda}(\gamma) = 0$ and γ has only one child.

Balanced partition of $\boldsymbol{V(T)}$. Recall that $\pi(T, \boldsymbol{\lambda}) = \sum_{u \in V(T)} \boldsymbol{\lambda}(u) \cdot d(u, \gamma(T))$. When the context is clear we simply write $\pi(T) = \pi(T, \boldsymbol{\lambda})$, and for a connected subset $U \subseteq V(T)$ we simply write $\pi(U) := \pi(T[U], \boldsymbol{\lambda}|_U)$.

▶ **Definition 18.** *Given a stochastic instance $(T, \boldsymbol{\lambda})$, we say that $U \subseteq V(T)$ is* balanced *if U is connected and if one of the following conditions holds:*

(1) *U is of* type-I*: $\pi(U) \leq 1$, and either $\gamma(U) = \gamma(T)$ or $\pi(U \cup \{par(\gamma(U))\}) > 1$;*

(2) *U is of* type-II*: $\pi(U) > 1$, and for each child vertex y of $\gamma(U)$ in $T[U]$, we have $\pi(\{\gamma(U)\} \cup (U \cap V_y)) < 1$.*

Remark that the root $\gamma(U)$ of a balanced type-II part U must have at least two children in $T[U]$.

▶ **Definition 19.** *Given a stochastic instance $(T, \boldsymbol{\lambda})$ and a partition $\mathcal{P}$ of the vertices $V(T)$, we say that $\mathcal{P}$ is a* balanced partition *of tree T if every part $U \in \mathcal{P}$ is balanced.*

See Figure 2 for an example of a balanced partition. If $\mathcal{P}$ is a *balanced partition* of T, then the part $U \in \mathcal{P}$ containing $\gamma(T)$ is called the *root part* in $\mathcal{P}$. Since we assume that $\gamma(T)$ has only one child vertex, we deduce from the previous remark that the root part is necessarily of type-I. Given a balanced partition $\mathcal{P}$, we denote $\mathcal{P}^* := \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$; $\mathcal{P}_1 \subseteq \mathcal{P}$ the set of type-I parts; $\mathcal{P}_1^* := \mathcal{P}_1 \cap \mathcal{P}^*$ and $\mathcal{P}_2 \subseteq \mathcal{P}$ the set of type-II parts.

▶ **Lemma 20.** *Given any stochastic instance $(T, \boldsymbol{\lambda})$, there exists a balanced partition of T. Moreover, such a partition can be computed in $O(|V(T)|^2)$ time.*

The algorithm to construct a partition $\mathcal{P}$ works as follows. We order the vertices $u_1, \ldots, u_n$ by decreasing distances from the root, i.e., for $1 \leq i < j$, $d(u_i, \gamma(T)) \geq d(u_j, \gamma(T))$. Let $\mathcal{P}^{(0)} = \emptyset$. For each $i \in [n]$, let $C_i \subseteq \{1, \ldots, n-1\}$ be the subset of indexes j s.t. (i) u_j is a child of u_i; (ii) $u_j \notin \bigcup_{U \in \mathcal{P}^{(i-1)}} U$. Define $U_i := (\bigcup_{j \in C_i} U_j) \bigcup \{u_i\}$ recursively. If $i = n$ (i.e., if u_i is the root of T) or $\pi(U_i \cup \{\text{par}(u_i)\}) > 1$, then define $\mathcal{P}^{(i)} := \mathcal{P}^{(i-1)} \cup \{U_i\}$. Otherwise, define $\mathcal{P}^{(i)} := \mathcal{P}^{(i-1)}$. See Figure 3 for a visual support.

The heavy instance. Given a stochastic instance $(T, \boldsymbol{\lambda})$, and a balanced partition $\mathcal{P}$ of T, we construct a tree T' that we call the *augmented tree* of T. This tree is essentially a copy of T with additional one or two vertices for each part of $\mathcal{P}^*$.[4] Then, we define arrival rates $\boldsymbol{\lambda}^h$ on T' in a way that the stochastic instance $(T', \boldsymbol{\lambda}^h)$ is heavy. Finally, we construct from a request sequence σ, the corresponding *heavy sequence* σ^h for the augmented tree.

[4] Recall that $\mathcal{P}^* = \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$, where $\gamma(\mathcal{P})$ denotes the particular part in $\mathcal{P}$ including the tree root $\gamma(T)$.

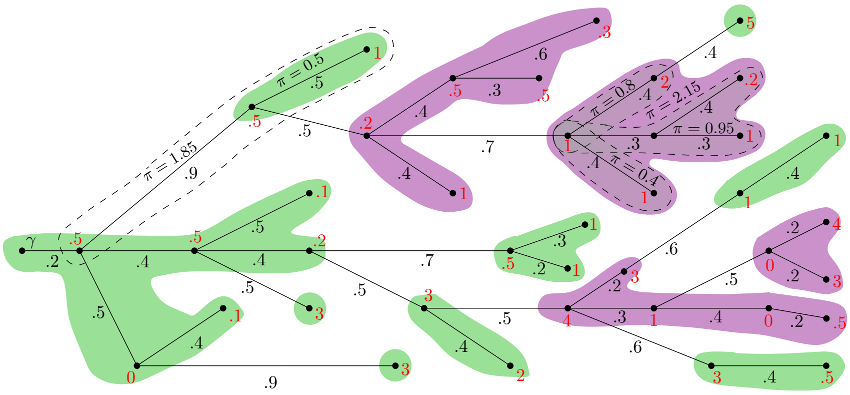


Figure 2 An example of a balanced partition (Definition 19). The weight of each edge is shown in black, and the arrival rate of each vertex is shown in red. Green subsets corresponds to parts of type-I while purple ones correspond to parts of type-II. Some value of π are shown for the top-left type-I part and for the top-right type-II part.

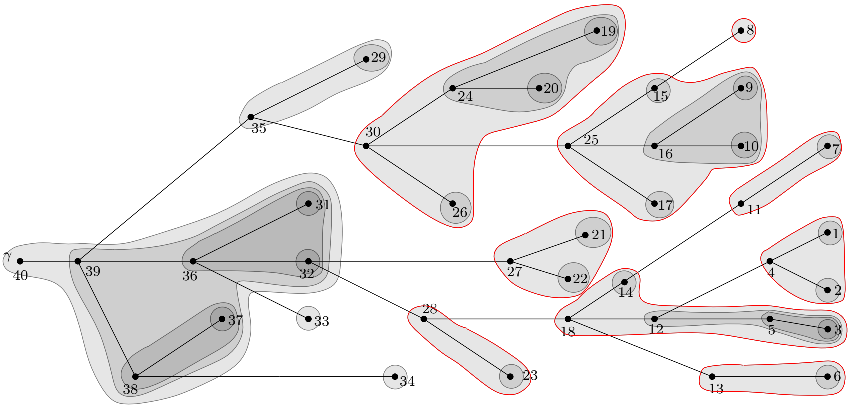


Figure 3 Construction of a balanced partition in the proof of Lemma 20. The weights of the edges and the arrival rates of the vertices are the same as in Figure 2. The numbers represent an ordering of the vertices. The gray sets corresponds to U_i, for $i \in [40]$. We illustrate the step $i = 30$ of the algorithm. We have $C_{30} = \{24, 26\}$ and $U_{30} = \{u_{30}\} \cup U_{24} \cup U_{30} = \{u_{30}, u_{24}, u_{19}, u_{20}, u_{26}\}$. Since $\pi(U_{30} \cup \{u_{25}\}) = 2.65 > 1$, we add U_{30} into $\mathcal{P}^{(29)}$ to create $\mathcal{P}^{(30)}$ (red stoke). We remark that U_{30} is a balanced subset of type-II and U_{27} is a balanced subset of type-II, while U_{25} is not a balanced subset.

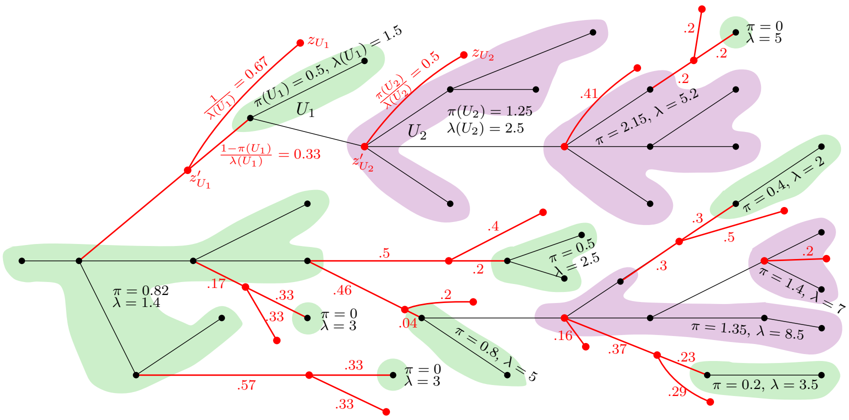


Figure 4 The construction of the augmented tree associated with the instance and the balanced partition of Figure 2. The new edges and vertices are shown in red. The illustrate the calculation of the length of these edges for a part U_1 of type-I and for a part U_2 of type-II. For each part U of the partition, we indicate the values of $\boldsymbol{\lambda}(U)$ and $\pi(U)$. For simplicity, we have rounded the values to their second decimal.

To construct the augmented tree, define $T' = (V', E')$ where $V' = V(T) \cup \{z_U, z'_U : U \in \mathcal{P}^*\}$, and the edge set E' is constructed based on $E(T)$ as follows. First, for each $U \in \mathcal{P}_1^*$, replace the edge $(\gamma(U), \mathrm{par}(\gamma(U)))$ of length $w_{\gamma(U)}$ by two edges $(\gamma(U), z'_U)$ and $(z'_U, \mathrm{par}(\gamma(U)))$ of respective lengths $(1 - \pi(U))/\boldsymbol{\lambda}(U)$ and $w_{\gamma(U)} - (1 - \pi(U))/\boldsymbol{\lambda}(U)$, where $\mathrm{par}(\gamma(U))$ denotes the parent of $\gamma(U)$ in T. Then, add an edge (z_U, z'_U) of weight $1/\boldsymbol{\lambda}(U)$. Finally, for each $U \in \mathcal{P}_2$, set $z'_U = \gamma(U)$, and add an edge (z_U, z'_U) of weight $\pi(U)/\boldsymbol{\lambda}(U)$.

This completes the construction of the augmented tree (see Figure 4 for visual support). Note that if a part $U = \{u\}$ in $\mathcal{P}$ contains only one vertex, then we have $\pi(U) = 0$, and thus part U is necessarily of type-I. To simplify, in the following, we identify vertices in T with their copy in T' and consider that $V(T)$ is a subset of $V(T')$. For the arrival rates of the heavy instance, recall that $\mathcal{P}^* = \mathcal{P} \setminus \{\gamma(\mathcal{P})\}$, where $\gamma(\mathcal{P})$ denotes the part in $\mathcal{P}$ containing the root $\gamma(T)$. We define $\boldsymbol{\lambda}^h : V(T') \to \mathbb{R}_+$ as follows: for each $U \in \mathcal{P}^*$, set $\boldsymbol{\lambda}^h(z_U) = \boldsymbol{\lambda}(U)$; and $\boldsymbol{\lambda}^h(u) = 0$ otherwise.[5]

▶ **Definition 21.** *Given a stochastic instance $(T, \boldsymbol{\lambda})$, a balanced partition $\mathcal{P}$ of tree T, the corresponding augmented tree T', and a sequence of request $\sigma \sim (T, \boldsymbol{\lambda})^\tau$, we construct the* heavy sequence *associated with σ for T' and denoted by σ^h as follows: for each request $r = (u, t) \in \sigma$ located on some part $U \in \mathcal{P}^*$ (i.e., $u \in U$), there is a request (z_U, t) in σ^h.*

The algorithm GEN. The input is a stochastic instance $(T, \boldsymbol{\lambda})$, known in advance, and a sequence of requests σ for T, revealed over time. In the pre-processing step, GEN computes a balanced partition $\mathcal{P}$ of T (Lemma 20), a light instance $(T[\gamma(\mathcal{P})], \sigma|_{\gamma(\mathcal{P})})$, and the heavy instance (T', σ^h). At each request arrival, GEN updates the sequences of requests $\sigma|_{\gamma(\mathcal{P})}$ and σ^h. The algorithm runs PLAN (Algorithm 1) on input (T', σ^h). Suppose that PLAN serves at time t a set of vertices $\{z_U, U \in \mathcal{P}'\} \subseteq V(T')$ for some subset $\mathcal{P}' \subseteq \mathcal{P}^*$. Then, GEN serves at time t all pending requests on vertices $\left(\bigcup_{U \in \mathcal{P}'} U\right) \subseteq V(T)$.

[5] It is important to notice that σ^h can be constructed in an online fashion: for any time t, the restriction of the σ^h to the requests that arrives before t only depends on the requests that arrives before t in σ.

In parallel, the algorithm runs INSTANT on input $(T[\gamma(\mathcal{P})], \sigma|_{\gamma(\mathcal{P})})$, and performs the same services. This finishes the description of the algorithm GEN see the formal pseudo-code of GEN (Algorithm 2) and a visual support in Figures 2, 3 and 4. The detailed proof of GEN achieving a constant ratio of expectations can be found in the appendix of the full version [57].

Algorithm 2 GEN.

```
Input: stochastic instance $(T, \boldsymbol{\lambda})$ and $\sigma \sim (T, \boldsymbol{\lambda})^\tau$
Output: a valid schedule of $\sigma$
– – pre-processing the given instance ——
produce a balanced partition $\mathcal{P}$ for $T$ (see Lemma 20);
construct the heavy instance $(T', \boldsymbol{\lambda}^h)$;
use PLAN (Algorithm 1) to determine the period of the vertices of $T'$;
—— Serve the requests ——
for each request $r \in \sigma$ do
    if $r$ arrives in $\gamma(\mathcal{P})$ then
        serve $r$ immediately.
    end
    if $r$ arrives in a vertex of $U \in \mathcal{P}^*$ then
        serve $r$ at time $t(r')$ where $r' \in \sigma^h$ is the corresponding request located on $z_U$
          and $t(r')$ is the time at which $r'$ is served by PLAN($\sigma^h$).
    end
end
```

7 Other related works

The MLA problem was first introduced by Bienkowski et al. [16] and they study a more general version in their paper, where the cost of delaying a request r by a duration t is $f_r(t)$. Bienkowski et al. proposed an $O(d^4 2^d)$-competitive online algorithm for this general delay cost version problem, where d denotes the depth of the given tree. A deadline version of MLA is also considered in [16], where each request r has a time window (between its arrival and its deadline) and it has to be served no later than its deadline. The target is to minimize the total service cost for serving all the requests. For this deadline version problem, they proposed an online algorithm with a better competitive ratio of $d^2 2^d$. Later, the competitiveness of MLA was further improved to $O(d^2)$ [13] for the general delay cost version and to $O(d)$ [27, 58] for the deadline version. However, for the delay cost version, no matching lower bound has been found thus far – the current best lower bound on MLA with delays is only 4 [16, 17, 18], restricted to a path case with linear delays. In the offline setting, MLA is NP-hard in both delay and deadline versions [3, 15], and a 2-approximation algorithm was proposed by Becchetti et al. [15] for the deadline version. For a special path case of MLA with the linear delay, Bienkowski et al. [22] proved that the competitiveness is between 3.618 and 5, improving on an earlier 8-competitive algorithm given by Brito et al. [26]. Thus far, no previous work has studied MLA in the stochastic input model, no matter the delay or deadline versions.

Two special cases of MLA with linear delays, one called TCP-acknowledgment ($d = 1$) and one called Joint Replenishment (abbr. JRP, $d = 2$) are of particular interests: TCP-acknowledgment (a.k.a. single item lot-sizing problem, [25, 41, 61, 29, 44]) models the data transmission issue from sensor networks [68, 51], while JRP models the inventory control issue from supply chain management [5, 37, 42, 64, 48]. For TCP-acknowledgment, in the online setting there exists an optimal 2-competitive deterministic algorithm [32] and an optimal $e/(e-1)$-competitive randomized algorithm [45, 63]; in the offline setting, the problem can be solved in $\mathrm{O}(n \log n)$ time, n denoting the number of requests [1]. For JRP, the competitiveness is between 3 [28] and 2.754 [21]; in the offline setting, JRP is NP-hard [3] and also APX-hard [60, 20]. The current best approximation ratio for JRP is 1.791 [53, 54, 52, 21]. For a deadline version of JRP, Bienkowski et al. [21] proposed an optimal 2-competitive algorithm.

Another problem, called online service with delays (OSD), first introduced by Azar et al. [9], is closely related to MLA (with linear delays). In this OSD problem, a n-points metric space is given as input. The requests arrive at metric points over time, and a server is available to serve the requests. The target is to serve all the requests in an online manner such that their total delay cost plus the total distance traveled by the server is minimized. Note that MLA can be seen as a special case of OSD when the given metric is a tree, and the server has to always come back to a particular tree vertex immediately after serving some requests elsewhere. For OSD, Azar et al. [9] proposed an $\mathrm{O}(\log^4 n)$-competitive online algorithm in their paper. Later, the competitive ratio for OSD is improved from $O(\log^2 n)$ (by Azar and Touitou [13]) to $O(\log n)$ (by Touitou [67]).

Recently, many other online problems with delays/deadline have also drawn a lot of attention besides MLA, such as online matching with delays [33, 6, 4, 24, 23, 34, 10, 31, 55, 12, 59, 56, 49], online service with delays [9, 13, 66, 67], facility location with delays/deadline [19, 13, 14], Steiner tree with delays/deadline [14], bin packing with delays [8, 35, 36, 2], set cover with delays [7, 65, 50], paging with delays/deadline [38, 39], list update with delays/deadline [11], and many others [59, 30, 66, 40, 43, 46].

8 Concluding remarks

In this paper, we studied MLA with additional stochastic assumptions on the sequence of the input requests. In the following, we briefly discuss some potential future directions.

Does the greedy algorithm achieve a constant ratio of expectations? An intuitive heuristic algorithm for MLA is *Greedy*, which works as follows: *each time when a set of requests R arriving at vertices $U \subseteq V(T)$ have the total delay cost equal to the weight of the minimal subtree of T including γ and U, serve all the requests R.* Does this greedy algorithm achieve a constant ratio of expectations?

Is it possible to generalize MLA with edge capacity and k tree roots? One practical scenario on MLA is that each edge has a capacity on the maximum number of requests served in one service if this edge is used, such as [61, 44, 64]. We conjecture that some $\mathrm{O}(1)$-RoE online algorithm can be proposed for this generalized MLA with edge capacity. Another generalized version of MLA is to assume k tree roots available for serving requests concurrently. That is, a set of pending requests can be served together by connecting to any of k servers. The question is, how to design an online algorithm for this k-MLA problem? Does there exist $\mathrm{O}(1)$-RoE algorithm still?

What about the other online network design problems with delays in the Poisson arrival model? Recall that the online problems of service with delays (and its generalization called k-services with delays), facility location with delays, Steiner tree/forest with delays are all closely related to MLA. Does there exist online algorithm with $\mathrm{O}(1)$-RoE for each problem?

References

1 Alok Aggarwal and James K. Park. Improved algorithms for economic lot size problems. *Operations research*, 41(3):549–571, 1993. doi:10.1287/OPRE.41.3.549.

2 Lauri Ahlroth, André Schumacher, and Pekka Orponen. Online bin packing with delay and holding costs. *Operations Research Letters*, 41(1):1–6, 2013. doi:10.1016/J.ORL.2012.10.006.

3 Esther Arkin, Dev Joneja, and Robin Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations research letters*, 8(2):61–66, 1989.

4 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Proc. APPROX / RANDOM*, pages 1:1–1:20, 2017. doi:10.4230/LIPICS.APPROX-RANDOM.2017.1.

5 Y. Askoy and S. S. Erenguk. Multi-item inventory models with coordinated replenishment: a survey. *International Journal of Operations and Production Management*, 8:63–73, 1988.

6 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proc. SODA*, pages 1051–1061, 2017. doi:10.1137/1.9781611974782.67.

7 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay–clairvoyance is not required. In *Proc. ESA*, pages 8:1–8:21, 2020. doi:10.4230/LIPICS.ESA.2020.8.

8 Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packing with delays. In *Proc. SPAA*, pages 1–10, 2019. doi:10.1145/3323165.3323180.

9 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proc, STOC*, pages 551–563, 2017. doi:10.1145/3055399.3055475.

10 Yossi Azar and Amit Jacob-Fanani. Deterministic min-cost matching with delays. *Theory of Computing Systems*, 64(4):572–592, 2020. doi:10.1007/S00224-019-09963-7.

11 Yossi Azar, Shahar Lewkowicz, and Danny Vainstein. List update with delays or time windows. In *Proc. ICALP*, pages 15:1–15:20, 2024. doi:10.4230/LIPICS.ICALP.2024.15.

12 Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In *Proc. SODA*, pages 301–320, 2021. doi:10.1137/1.9781611976465.20.

13 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proc. FOCS*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

14 Yossi Azar and Noam Touitou. Beyond tree embeddings–a deterministic framework for network design with deadlines or delay. In *Proc. FOCS*, pages 1368–1379, 2020. doi:10.1109/FOCS46700.2020.00129.

15 Luca Becchetti, Alberto Marchetti-Spaccamela, Andrea Vitaletti, Peter Korteweg, Martin Skutella, and Leen Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms*, 6(1):1–20, 2009. doi:10.1145/1644015.1644028.

16 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarcznỳ, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. Online algorithms for multi-level aggregation. In *Proc. ESA*, pages 12:1–12:17, 2016.

17 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarcznỳ, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. Online algorithms for multilevel aggregation. *Operations Research*, 68(1):214–232, 2020. doi:10.1287/OPRE.2019.1847.

18 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarcznỳ, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. New results on multi-level aggregation. *Theoretical Computer Science*, 861:133–143, 2021. doi:10.1016/J.TCS.2021.02.016.

19 Marcin Bienkowski, Martin Böhm, Jarosław Byrka, and Jan Marcinkowski. Online facility location with linear delay. In *Proc. APPROX/RANDOM*, pages 45:1–45:17, 2022. doi:10.4230/LIPICS.APPROX/RANDOM.2022.45.

20 Marcin Bienkowski, Jarosław Byrka, Marek Chrobak, Neil Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Świrszcz, and Neal E. Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015. doi:10.1007/S10951-014-0392-Y.

21 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. SODA*, pages 42–54, 2014.

22 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. WADS*, pages 133–145, 2013.

23 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Paweł Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Proc. WAOA*, pages 51–68, 2018.

24 Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Proc. WAOA*, pages 132–146, 2017.

25 Nadjib Brahimi, Stéphane Dauzere-Peres, Najib M Najid, and Atle Nordli. Single item lot sizing problems. *European Journal of Operational Research*, 168(1):1–16, 2006. doi:10.1016/J.EJOR.2004.01.054.

26 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64:584–605, 2012. doi:10.1007/S00453-011-9567-5.

27 Niv Buchbinder, Moran Feldman, Joseph Naor, and Ohad Talmon. O (depth)-competitive algorithm for online multi-level aggregation. In *Proc. SODA*, pages 1235–1244, 2017.

28 Niv Buchbinder, Tracy Kimbrelt, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proc. SODA*, pages 952–961, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347186.

29 Maxim A. Bushuev, Alfred Guiffrida, M.Y. Jaber, and Mehmood Khan. A review of inventory lot sizing review papers. *Management Research Review*, 38(3):283–298, 2015.

30 Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In *Proc. ICALP*, 2022.

31 Lindsey Deryckere and Seeun William Umboh. Online matching with set and concave delays. In *Proc. APPROX/RANDOM*, pages 17:1–17:17, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.17.

32 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the tcp acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001. doi:10.1145/375827.375843.

33 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proc. STOC*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

34 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. *Theoretical Computer Science*, 754:122–129, 2019. doi:10.1016/J.TCS.2018.07.004.

35 Leah Epstein. On bin packing with clustering and bin packing with delays. *Discrete Optimization*, 41:100647, 2021. doi:10.1016/J.DISOPT.2021.100647.

36 Leah Epstein. Open-end bin packing: new and old analysis approaches. *Discrete Applied Mathematics*, 321:220–239, 2022. doi:10.1016/J.DAM.2022.07.003.

37 Suresh K. Goyal and Ahmet T. Satir. Joint replenishment inventory control: deterministic and stochastic models. *European journal of operational research*, 38(1):2–13, 1989.

38 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In *Proc. STOC*, pages 1125–1138, 2020. doi:10.1145/3357713.3384277.

39 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k-server and an extension to time-windows. In *Proc. FOCS*, pages 504–515, 2022.

40 Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. In *Proc. INFOCOM*, pages 1–10, 2023. doi:10.1109/INFOCOM53939.2023.10228882.

41 Raf Jans and Zeger Degraeve. Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 46(6):1619–1643, 2008.

42 Dev Joneja. The joint replenishment problem: new heuristics and worst case performance bounds. *Operations Research*, 38(4):711–723, 1990. doi:10.1287/OPRE.38.4.711.

43 Naonori Kakimura and Tomohiro Nakayoshi. Deterministic primal-dual algorithms for online k-way matching with delays. In *Proc. ICCC*, pages 238–249, 2023. doi:10.1007/978-3-031-49193-1_18.

44 Behrooz Karimi, S.M.T. Fatemi Ghomi, and J.M. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003.

45 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proc. STOC*, pages 502–509, 2001. doi:10.1145/380752.380845.

46 Yasushi Kawase and Tomohiro Nakayoshi. Online matching with delays and size-based costs. *arXiv preprint arXiv:2408.08658*, 2024. doi:10.48550/arXiv.2408.08658.

47 Sanjeev Khanna, Joseph Seffi Naor, and Dan Raz. Control message aggregation in group communication protocols. In *Proc. ICALP*, pages 135–146, 2002.

48 Moutaz Khouja and Suresh Goyal. A review of the joint replenishment problem literature: 1989–2005. *European journal of operational Research*, 186(1):1–16, 2008. doi:10.1016/J.EJOR.2007.03.007.

49 Tung-Wei Kuo. Online deterministic minimum cost bipartite matching with delays on a line. *arXiv preprint*, 2024. doi:10.48550/arXiv.2408.02526.

50 Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In *Proc. SODA*, pages 1594–1610, 2023. doi:10.1137/1.9781611977554.CH59.

51 Ka-Cheong Leung, Victor OK Li, and Daiqin Yang. An overview of packet reordering in transmission control protocol (tcp): problems, solutions, and challenges. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):522–535, 2007. doi:10.1109/TPDS.2007.1011.

52 Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008. doi:10.1287/MNSC.1070.0781.

53 Retsef Levi, Robin Roundy, and David B Shmoys. Primal-dual algorithms for deterministic inventory problems. In *Proc. STOC*, pages 353–362, 2004. doi:10.1145/1007352.1007410.

54 Retsef Levi and Maxim Sviridenko. Improved approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. APPROX-RANDOM*, pages 188–199, 2006. doi:10.1007/11830924_19.

55 Xingwu Liu, Zhida Pan, Yuyi Wang, and Roger Wattenhofer. Impatient online matching. In *Proc. ISAAC*, volume 123, pages 62:1–62:12, 2018. doi:10.4230/LIPICS.ISAAC.2018.62.

56 Mathieu Mari, Michał Pawłowski, Runtian Ren, and Piotr Sankowski. Online matching with delays and stochastic arrival times. In *Proc. AAMAS*, pages 976–984, 2023.

57 Mathieu Mari, Michał Pawłowski, Runtian Ren, and Piotr Sankowski. Online multi-level aggregation with delays and stochastic arrivals. *arXiv preprint arXiv:2404.09711*, 2024.

58 Jeremy McMahan. A d-competitive algorithm for the multilevel aggregation problem with deadlines. *arXiv preprint arXiv:2108.04422*, 2021. arXiv:2108.04422.

59 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Online k-way matching with delays and the h-metric. *arXiv preprint arXiv:2109.06640*, 2021. arXiv:2109.06640.

60 Tim Nonner and Alexander Souza. Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications*, 1(02):153–173, 2009. doi:10.1142/S1793830909000130.

61 Daniel Quadt and Heinrich Kuhn. Capacitated lot-sizing with extensions: a review. *Operation Research*, 6(1):61–83, 2008. doi:10.1007/S10288-007-0057-1.

62 Sheldon M. Ross. *Stochastic processes*, volume 2. Wiley New York, 1996.

63 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. STOC*, pages 592–601, 2000. doi:10.1145/335305.335385.

64 Sombat Sindhuchao, H. Edwin Romeijn, Elif Akçali, and Rein Boondiskulchok. An integrated inventory-routing system for multi-item joint replenishment with limited vehicle capacity. *Journal of Global Optimization*, 32:93–118, 2005. doi:10.1007/S10898-004-5908-0.

65 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In *Proc. ISAAC*, pages 53:1–53:16, 2021. doi:10.4230/LIPICS.ISAAC.2021.53.

66 Noam Touitou. Frameworks for nonclairvoyant network design with deadlines or delay. In *Proc. ICALP*, pages 105:1–105:20, 2023. doi:10.4230/LIPICS.ICALP.2023.105.

67 Noam Touitou. Improved and deterministic online service with deadlines or delay. In *Proc. STOC*, pages 761–774, 2023. doi:10.1145/3564246.3585107.

68 Wei Yuan, Srikanth V. Krishnamurthy, and Satish K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *Proc. GLOBECOM*, volume 1, pages 221–225, 2003. doi:10.1109/GLOCOM.2003.1258234.

On the Parameterized Complexity of Diverse SAT

Neeldhara Misra
Department of Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India

Harshil Mittal
Department of Computer Science and Engineering,
Indian Institute of Technology Gandhinagar, India

Ashutosh Rai
Department of Mathematics, Indian Institute of Technology Delhi,
New Delhi, India

Abstract

We study the Boolean Satisfiability problem (SAT) in the framework of diversity, where one asks for multiple solutions that are mutually far apart (i.e., sufficiently dissimilar from each other) for a suitable notion of distance/dissimilarity between solutions. Interpreting assignments as bit vectors, we take their Hamming distance to quantify dissimilarity, and we focus on the problem of finding two solutions. Specifically, we define the problem Max Differ SAT (resp. Exact Differ SAT) as follows: Given a Boolean formula ϕ on n variables, decide whether ϕ has two satisfying assignments that differ on at least (resp. exactly) d variables. We study the classical and parameterized (in parameters d and $n - d$) complexities of Max Differ SAT and Exact Differ SAT, when restricted to some classes of formulas on which SAT is known to be polynomial-time solvable. In particular, we consider affine formulas, Krom formulas (i.e., 2-CNF formulas) and hitting formulas.

For affine formulas, we show the following: Both problems are polynomial-time solvable when each equation has at most two variables. Exact Differ SAT is NP-hard, even when each equation has at most three variables and each variable appears in at most four equations. Also, Max Differ SAT is NP-hard, even when each equation has at most four variables. Both problems are W[1]-hard in the parameter $n - d$. In contrast, when parameterized by d, Exact Differ SAT is W[1]-hard, but Max Differ SAT admits a single-exponential FPT algorithm and a polynomial-kernel. For Krom formulas, we show the following: Both problems are polynomial-time solvable when each variable appears in at most two clauses. Also, both problems are W[1]-hard in the parameter d (and therefore, it turns out, also NP-hard), even on monotone inputs (i.e., formulas with no negative literals). Finally, for hitting formulas, we show that both problems can be solved in polynomial-time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Diverse solutions, Affine formulas, 2-CNF formulas, Hitting formulas

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.50

Funding *Neeldhara Misra*: Supported by IIT Gandhinagar and the SERB ECR grant ECR/2018/002967.
Harshil Mittal: Supported by IIT Gandhinagar and Google as an ISAAC sponsor.
Ashutosh Rai: Supported by IIT Delhi.

Acknowledgements We thank Saraswati Girish Nanoti for helpful discussions.

1 Introduction

We initiate a study of the problem of finding two satisfying assignments to an instance of SAT, with the goal of maximizing the number of variables that have different truth values under the two assignments, in the parameterized setting. This question is motivated by the broader framework of finding "diverse solutions" to optimization problems. When a

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 50; pp. 50:1–50:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

real-world problem is modelled as a computational problem, some contextual side-information is often lost. So, while two solutions may be equally good for the theoretical formulation, one of them may be better than the other for the actual practical application. A natural fix is to provide multiple solutions (instead of just one solution) to the user, who may then pick the solution that best fulfills her/his need. However, if the solutions so provided are all quite similar to each other, they may exhibit almost identical behaviours when judged on the basis of any relevant external factor. Thus, to ensure that the user is able to meaningfully compare the given solutions and hand-pick one of them, she/he must be provided a collection of *diverse solutions*, i.e., a few solutions that are sufficiently dissimilar from each other. This framework of *diversity* was proposed by Baste et. al. [3]. Since the late 2010s, several graph-theoretic and matching problems have been studied in this setting from an algorithmic standpoint. These include diverse variants of vertex cover [4], feedback vertex set [4], hitting set [4], perfect/maximum matching [17], stable matching [20], weighted basis of matroid [18], weighted common independent set of matroids [18], minimum s-t cut [11], spanning tree [22] and non-crossing matching [33].

The Boolean Satisfiability problem (SAT) asks whether a given Boolean formula has a satisfying assignment. This problem serves a crucial role in complexity theory [27], cryptography [32] and artificial intelligence [37]. In the early 1970s, SAT became the first problem proved to be NP-complete in independent works of Cook [8] and Levin [30]. Around the same time, Karp [27] built upon this result by showing NP-completeness of twenty-one graph-theoretic and combinatorial problems via reductions from SAT. In the late 1970s, Schaefer [35] formulated the closely related Generalized Satisfiability problem (SAT(S)), where each constraint applies on some variables, and it forces the corresponding tuple of their truth-values to belong to a certain Boolean relation from a fixed finite set S. His celebrated dichotomy result listed six conditions such that SAT(S) is polynomial-time solvable if S meets one of them; otherwise, SAT(S) is NP-complete.

Since SAT is NP-complete, it is unlikely to admit a polynomial-time algorithm, unless $\mathsf{P} = \mathsf{NP}$. Further, in the late 1990s, Impaglliazo and Paturi [25] conjectured that SAT is unlikely to admit even sub-exponential time algorithms, often referred to as the *exponential-time hypothesis*. To cope with the widely believed hardness of SAT, several special classes of Boolean formulas have been identified for which SAT is polynomial-time solvable. In the late 1960s, Krom [29] devised a quadratic-time algorithm to solve SAT on 2-CNF formulas. In the late 1970s, follow-up works of Even et. al. [16] and Aspvall et. al. [2] proposed linear-time algorithms to solve SAT on 2-CNF formulas. These algorithms used limited back-tracking and analysis of the strongly-connected components of the implication graph respectively. In the late 1980s, Iwama [26] introduced the class of hitting formulas, for which he gave a closed-form expression to count the number of satisfying assignments in polynomial-time. It is also known that SAT can be solved in polynomial-time on affine formulas using Gaussian elimination [21]. Some other polynomial-time recognizable classes of formulas for which SAT is polynomial-time solvable include Horn formulas [12, 36], CC-balanced formulas [7], matched formulas [19], renamable-Horn formulas [31] and q-Horn DNF formulas [5, 6].

Diverse variant of SAT. In this paper, we undertake a complexity-theoretic study of SAT in the framework of diversity. We focus on the problem of finding a *diverse pair of satisfying assignments* of a given Boolean formula, and we take the number of variables on which the two assignments differ as a measure of dissimilarity between them. Specifically, we define the problem Max Differ SAT (resp. Exact Differ SAT) as follows: Given a Boolean formula ϕ on n variables and a non-negative integer d, decide whether there are two satisfying assignments of ϕ that differ on at least d (resp. exactly d) variables. That is,

Table 1 Classical and parameterized (in parameters d and $n-d$) complexities of EXACT DIFFER SAT, when restricted to affine formulas, 2-CNF formulas and hitting formulas.

	Classical complexity	Parameter d	Parameter $n-d$
Affine formulas	NP-hard, even on $(3,4)$-affine formulas (Theorem 1) Polynomial-time on 2-affine formulas (Theorem 3)	W[1]-hard (Theorem 4)	W[1]-hard (Theorem 7)
2-CNF formulas	Polynomial-time on $(2,2)$-CNF formulas (Theorem 9)	W[1]-hard (Theorem 10)	?
Hitting formulas	Polynomial-time (Theorem 11)	–	–

Table 2 Classical and parameterized (in parameters d and $n-d$) complexities of MAX DIFFER SAT, when restricted to affine formulas, 2-CNF formulas and hitting formulas.

	Classical complexity	Parameter d	Parameter $n-d$
Affine formulas	NP-hard, even on 4-affine formulas (Theorem 2) Polynomial-time on 2-affine formulas (Theorem 3)	Single-exponential FPT (Theorem 5) Polynomial kernel (Theorem 6)	W[1]-hard (Theorem 7)
2-CNF formulas	Polynomial-time on $(2,2)$-CNF formulas (Theorem 8)	W[1]-hard (Theorem 10)	?
Hitting formulas	Polynomial-time (Theorem 11)	–	–

this problem asks whether there are two satisfying assignments of ϕ that overlap on at most $n-d$ (resp. exactly $n-d$) variables. Note that SAT can be reduced to its diverse variant by setting d to 0. Thus, as SAT is NP-hard in general, so is MAX/EXACT DIFFER SAT. So, it is natural to study the diverse variant on those classes of formulas for which SAT is polynomial-time solvable. In particular, we consider affine formulas, 2-CNF formulas and hitting formulas. We refer to the corresponding restrictions of MAX/EXACT DIFFER SAT as MAX/EXACT DIFFER AFFINE-SAT, MAX/EXACT DIFFER 2-SAT and MAX/EXACT DIFFER HITTING-SAT respectively. We analyze the classical and parameterized (in parameters d and $n-d$) complexities of these problems.

Related work. This paper is not the first one to study algorithms to determine the maximum number of variables on which two solutions of a given SAT instance can differ. Several exact exponential-time algorithms are known to find a pair of maximally far-apart satisfying assignments. In the mid 2000s, Angelsmark and Thapper [1] devised an $\mathcal{O}(1.7338^n)$ time algorithm to solve MAX HAMMING DISTANCE 2-SAT. Their algorithm involved a careful analysis of the micro-structure graph and used a solver for weighted 2-SAT as a subroutine. Around the same time, Dahllöf [10] proposed an $\mathcal{O}(1.8348^n)$ time algorithm for MAX HAMMING DISTANCE XSAT. In the late 2010s, follow-up works of Hoi et. al. [24, 23] developed algorithms for the same problem with improved running times, i.e., $\mathcal{O}(1.4983^n)$ for the general case, and $\mathcal{O}(1.328^n)$ for the case when every clause has at most three literals.

Parameterized complexity. In the 1990s, Downey and Fellows [14] laid the foundations of parameterized algorithmics. This framework measures the running time of an algorithm as a function of both the input size and a *parameter* k, i.e., a suitably chosen attribute of the input. Such a fine-grained analysis helps to cope with the lack of polynomial-time algorithms for NP-hard problems by instead looking for an algorithm with running time whose super-polynomial explosion is confined to the parameter k alone. That is, such an algorithm has a running time of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where $f(\cdot)$ is any computable function (could be exponential, or even worse) and n denotes the input size. Such an algorithm is said to be *fixed-parameter tractable* (FPT) because its running time is polynomially-bounded for every fixed value of the parameter k. For more on this paradigm, see [9].

Our findings. We summarize our findings in Table 1 and Table 2. In Section 3, we show that

- Exact Differ Affine-SAT is NP-hard, even on $(3, 4)$-affine formulas,
- Max Differ Affine-SAT is NP-hard, even on 4-affine formulas,
- Exact/Max Differ Affine-SAT is polynomial-time solvable on 2-affine formulas,
- Exact Differ Affine-SAT is W[1]-hard in the parameter d,
- Max Differ Affine-SAT admits a single-exponential FPT algorithm in the parameter d,
- Max Differ Affine-SAT admits a polynomial kernel in the parameter d, and
- Exact/Max Differ Affine-SAT is W[1]-hard in the parameter $n - d$.

In Section 4, we show that Exact/Max Differ 2-SAT can be solved in polynomial-time on $(2, 2)$-CNF formulas, and Exact/Max Differ 2-SAT is W[1]-hard in the parameter d. In Section 5, we show that Exact/Max Differ Hitting-SAT is polynomial-time solvable.

2 Preliminaries

A Boolean variable can take one of the two truth values: 0 (False) and 1 (True). We use n to denote the number of variables in a Boolean formula ϕ. An *assignment* of ϕ is a mapping from the set of all its n variables to $\{0, 1\}$. A *satisfying assignment* of ϕ is an assignment σ such that ϕ evaluates to 1 under σ, i.e., when every variable x is substituted with its assigned truth value $\sigma(x)$. We say that two assignments σ_1 and σ_2 *differ* on a variable x if they assign different truth values to x. That is, one of them sets x to 0, and the other sets x to 1. Otherwise, we say that σ_1 and σ_2 *overlap* on x. That is, either both of them set x to 0, or both of them set x to 1.

A *literal* is either a variable x (called a *positive literal*) or its negation $\neg x$ (called a *negative literal*). A *clause* is a disjunction (denoted by $\bigvee$) of literals. A Boolean formula in *conjunctive normal form*, i.e., a conjunction (denoted by $\bigwedge$) of clauses, is called a *CNF formula*. A 2-*CNF formula* is a CNF formula with at most two literals per clause. A $(2, 2)$-*CNF formula* is a 2-CNF formula in which each variable appears in at most two clauses. An *affine formula* is a conjunction of linear equations over the two-element field $\mathbb{F}_2$. We use $\oplus$ to denote the XOR operator, i.e., addition-modulo-2. A 2-*affine formula* is an affine formula in which each equation has at most two variables. Similarly, a 3-*affine* (resp. 4-*affine*) *formula* is an affine formula in which each equation has at most three (resp. four) variables. A $(3, 4)$-*affine formula* is a 3-affine formula in which each variable appears in at most four equations.

The solution set of a system of linear equations can be obtained in polynomial-time using Gaussian elimination [21]. It may have no solution, a unique solution or multiple solutions. When it has multiple solutions, the solution set is described as follows: Some variables are allowed to take any value; we call them *free variables*. The remaining variables take values that are dependent on the values taken by the free variables; we call them *forced variables*. That is, the value taken by any forced variable is a linear combination of the values taken by some free variables. For example, consider the following system of three linear equations over

$\mathbb{F}_2$: $x \oplus y \oplus z = 1$, $u \oplus y = 1, w \oplus z = 1$. This system has multiple solutions, and its solution set can be described as $\{(x, y, z, u, w) \mid y \in \mathbb{F}_2, z \in \mathbb{F}_2, x = y \oplus z \oplus 1, u = y \oplus 1, w = z \oplus 1\}$. Here, y and z are free variables. The remaining variables, i.e., x, u and w, are forced variables.

A *hitting formula* is a CNF formula such that for any pair of its clauses, there is some variable that appears as a positive literal in one clause, and as a negative literal in the other clause. That is, no two of its clauses can be simultaneously falsified. Note that the number of unsatisfying assignments of a hitting formula ϕ on n variables can be expressed as follows:

$$\sum_{C\colon C \text{ is a clause of } \phi} |\{\sigma \mid \sigma \text{ is an assignment of } \phi \text{ that falsifies } C\}| = \sum_{C\colon C \text{ is a clause of } \phi} 2^{n-|\mathsf{vars}(C)|}$$

Here, we use $\mathsf{vars}(C)$ to denote the set of all variables that appear in the clause C.

We use the following as source problems in our reductions:

- Independent Set. Given a graph G and a positive integer k, decide whether G has an independent set of size k. This problem is known to be NP-hard on cubic graphs [34], and W[1]-hard in the parameter k [15].
- Multicolored Clique. Given a graph G whose vertex set is partitioned into k color-classes, decide whether G has a k-sized clique that picks exactly one vertex from each color-class. This problem is known to be NP-hard on r-regular graphs [9].
- Exact Even Set. Given a universe $\mathcal{U}$, a family $\mathcal{F}$ of subsets of $\mathcal{U}$ and a positive integer k, decide whether there is a set $X \subseteq \mathcal{U}$ of size exactly k such that $|X \cap S|$ is even for all sets S in the family $\mathcal{F}$. This problem is known to be W[1]-hard in the parameter k [13].
- Odd Set (resp. Exact Odd Set). Given a universe $\mathcal{U}$, a family $\mathcal{F}$ of subsets of $\mathcal{U}$ and a positive integer k, decide whether there is a set $X \subseteq \mathcal{U}$ of size at most k (resp. exactly k) such that $|X \cap S|$ is odd for all sets S in the family $\mathcal{F}$. Both these problems are known to be W[1]-hard in the parameter k [13].

We use a polynomial-time algorithm for the following problem as a sub-routine:

- Subset Sum problem. Given a multi-set of integers $\{w_1, \ldots, w_p\}$ and a target sum k, decide whether there exists $X \subseteq [p]$ such that $\sum_{i \in X} w_i = k$. This problem is known to be polynomial-time solvable when the input integers are specified in unary [28].

We use the notation $\mathcal{O}^\star(\cdot)$ to hide polynomial factors in running time.

3 Affine formulas

In this section, we focus on Exact Differ Affine-SAT, i.e, finding two different solutions to affine formulas. To begin with, we show that finding two solutions that differ on *exactly* d variables is hard even for $(3, 4)$-affine formulas: recall that these are instances where every equation has at most three variables and every variable appears in at most four equations.

▶ **Theorem 1.** *Exact Differ Affine-SAT is* NP*-hard, even on* $(3, 4)$*-affine formulas.*

Proof. We describe a reduction from Independent Set on Cubic graphs. Consider an instance (G, k) of Independent Set, where G is a cubic graph. We construct an affine formula ϕ as follows: For every vertex $v \in V(G)$, introduce a variable x_v, its $3k$ copies (say $x_v^1, \ldots, x_v^{3k}$), and $3k$ equations: $x_v \oplus x_v^1 = 0$, $x_v^1 \oplus x_v^2 = 0, \ldots, x_v^{3k-1} \oplus x_v^{3k} = 0$. For every edge $e = uv \in E(G)$, introduce variable y_e and equation $x_u \oplus x_v \oplus y_e = 0$. We set $d = k \cdot (3k + 4)$. For every vertex $v \in V(G)$, the variable x_v appears in four equations (i.e., $x_v \oplus x_v^1 = 0$ and the three equations corresponding to the three edges incident to v in G), each of $x_v^1, \ldots, x_v^{3k-1}$ appears in two equations, and x_v^{3k} appears in one equation. For every edge $e \in E(G)$, the variable y_e appears in one equation. So, overall, every variable appears in at most four equations. Also, the equation corresponding to any edge contains three variables, and the remaining equations contain two variables each. Therefore, ϕ is a $(3, 4)$-affine formula.

Now, we prove that (G, k) is a YES instance of Independent Set if and only if (ϕ, d) is a YES instance of Exact Differ Affine-SAT. At a high level, we argue this equivalence as follows: In the forward direction, we show that the two desired satisfying assignments are the all 0 assignment, and the assignment that i) assigns 1 to every x variable (and also, its $3k$ copies) that corresponds to a vertex of the independent set, ii) assigns 1 to every y variable that corresponds to an edge that has one endpoint inside the independent set and the other endpoint outside it, iii) assigns 0 to every x variable (and also, its $3k$ copies) that corresponds to a vertex outside the independent set, and iv) assigns 0 to every y variable that corresponds to an edge that has both its endpoints outside the independent set. In the reverse direction, we show that the desired k-sized independent set consists of those vertices that correspond to the x variables on which the two assignments differ. We now turn to a proof of equivalence.

Forward direction. Suppose that G has a k-sized independent set, say S. Let σ_1 and σ_2 be assignments of ϕ defined as follows: For every vertex $v \in V(G) \setminus S$, both σ_1 and σ_2 set $x_v, x_v^1, \ldots, x_v^{3k}$ to 0. For every vertex $v \in S$, σ_1 sets $x_v, x_v^1, \ldots, x_v^{3k}$ to 0, and σ_2 sets $x_v, x_v^1, \ldots, x_v^{3k}$ to 1. For every edge $e \in E(G)$ that has both its endpoints in $V(G) \setminus S$, both σ_1 and σ_2 set y_e to 0. For every edge $e \in E(G)$ that has one endpoint in S and the other endpoint in $V(G) \setminus S$, σ_1 sets y_e to 0, and σ_2 sets y_e to 1.

As σ_1 sets all variables to 0, it is clear that it satisfies ϕ. Now, we show that σ_2 satisfies ϕ. Consider any edge $e = uv \in E(G)$ and its corresponding equation $x_u \oplus x_v \oplus y_e = 0$. If both endpoints of e belong to $V(G) \setminus S$, then σ_2 sets x_u, x_v and y_e to 0. Also, if e has one endpoint (say u) in S, and the other endpoint in $V(G) \setminus S$, then σ_2 sets x_u to 1, x_v to 0 and y_e to 1. Therefore, in both cases, $x_u \oplus x_v \oplus y_e$ takes the truth value 0 under σ_2. Also, for any vertex $v \in V(G)$, since σ_2 gives the same truth value to $x_v, x_v^1, \ldots, x_v^{3k}$ (i.e., all 1 if $v \in S$, and all 0 if $v \in V(G) \setminus S$), it also satisfies the equations $x_v \oplus x_v^1 = 0, x_v^1 \oplus x_v^2 = 0, \ldots, x_v^{3k-1} \oplus x_v^{3k} = 0$. Thus, σ_2 is a satisfying assignment of ϕ.

As G is a cubic graph, every vertex in S is incident to three edges in G. Also, as S is an independent set, none of these edges has both endpoints in S. Therefore, there are $3 \cdot |S|$ edges that have one endpoint in S and the other endpoint in $V(G) \setminus S$. Note that σ_1 and σ_2 differ on the y variables that correspond to these $3 \cdot |S|$ edges. Also, they differ on $|S|$ many x variables, and their $3k \cdot |S|$ copies. Therefore, overall, they differ on $(3k+1) \cdot |S| + 3 \cdot |S| = k \cdot (3k+4)$ variables. Hence, (ϕ, d) is a YES instance of Exact Differ Affine SAT.

Reverse direction. Suppose that (ϕ, d) is a YES instance of Exact Differ Affine-SAT. That is, there exist satisfying assignments σ_1 and σ_2 of ϕ that differ on $k \cdot (3k+4)$ variables. Let $S := \{v \in V(G) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } x_v\}$. We show that S is a k-sized independent set of G. Let $e(S, \bar{S})$ denote the number of edges in G that have one endpoint in S and the other endpoint in $V(G) \setminus S$. Now, let us express the number of variables on which σ_1 and σ_2 differ in terms of $|S|$ and $e(S, \bar{S})$.

Consider any edge $e = uv \in E(G)$. First, suppose that e has both its endpoints in S. Then, as σ_1 and σ_2 differ on both x_u and x_v, the expression $x_u \oplus x_v$ takes the same truth value under σ_1 and σ_2. So, as both of them satisfy the equation $x_u \oplus x_v \oplus y_e = 0$, it follows that σ_1 and σ_2 must overlap on y_e. Next, suppose that e has both its endpoints in $V(G) \setminus S$. Then, as σ_1 and σ_2 overlap on both x_u and x_v, the expression $x_u \oplus x_v$ takes the same truth value under σ_1 and σ_2. So, again, σ_1 and σ_2 must overlap on y_e. Next, suppose that e has one endpoint (say u) in S and the other endpoint in $V(G) \setminus S$. Then, as σ_1 and σ_2 differ on x_u and overlap on x_v, the expression $x_u \oplus x_v$ takes different truth values under σ_1 and σ_2. So, as both σ_1 and σ_2 satisfy the equation $x_u \oplus x_v \oplus y_e = 0$, it follows that σ_1 and σ_2 must differ on y_e. So, overall, σ_1 and σ_2 differ on $e(S, \bar{S})$ many y variables.

For any vertex $v \in V(G)$, since any satisfying assignment satisfies the equations $x_v \oplus x_v^1 = 0, x_v^1 \oplus x_v^2 = 0, \ldots, x_v^{3k-1} \oplus x_v^{3k} = 0$, it must assign the same truth value to $x_v, x_v^1, \ldots, x_v^{3k}$. So, for any $v \in S$, as σ_1 and σ_2 differ on x_v, they also differ on $x_v^1, \ldots, x_v^{3k}$. Similarly, for any $v \in V(G) \setminus S$, as σ_1 and σ_2 overlap on x_v, they also overlap on $x_v^1, \ldots x_v^{3k}$. So, overall, σ_1 and σ_2 differ on $|S|$ many x variables and their $3k \cdot |S|$ copies. Now, summing up the numbers of y variables and x variables (and their copies) on which σ_1 and σ_2 differ, we get

$$e(S, \bar{S}) + (3k + 1) \cdot |S| = k \cdot (3k + 4) \tag{1}$$

Let $e(S, S)$ denote the number of edges in G that have both endpoints in S. Note that

$$\sum_{v \in S} \text{degree}_G(v) = 2 \cdot e(S, S) + e(S, \bar{S})$$

Also, as G is a cubic graph, we know that $\text{degree}_G(v) = 3$ for all $v \in S$. Therefore, we get $e(S, \bar{S}) = 3 \cdot |S| - 2 \cdot e(S, S)$. Putting this expression for $e(S, \bar{S})$ in Equation (1), we have

$$(3k + 4) \cdot \big(|S| - k\big) = 2 \cdot e(S, S) \tag{2}$$

If $|S| \geqslant k + 1$, then LHS of Equation (1) becomes $\geqslant (3k + 1) \cdot (k + 1) = k \cdot (3k + 4) + 1$, which is greater than its RHS. So, we must have $|S| \leqslant k$. Also, as RHS of Equation (2) is non-negative, so must be its LHS. This gives us $|S| \geqslant k$. Therefore, it follows that $|S| = k$. Putting $|S| = k$ in Equation (2), we also get $e(S, S) = 0$. That is, S is an independent set of G. Hence, (G, k) is a YES instance of Independent Set.

This proves Theorem 1. ◀

We now turn to Max Differ Affine-SAT, i.e, finding two solutions that differ on *at least* d variables. We show that this is hard for affine formulas of bounded arity.

▶ **Theorem 2.** *Max Differ Affine-SAT is* NP*-hard, even on* 4*-affine formulas.*

Proof. We describe a reduction from Multicolored Clique on Regular graphs. Consider an instance (G, k) of Multicolored Clique, where G is a r-regular graph. We assume that each color-class of G has size $N := 2 \cdot 3^q$. It can be argued that a suitably-sized r-regular graph exists whose addition to the color-class makes this assumption hold true. We construct an affine formula ϕ as follows: For every vertex $v \in V(G)$, introduce a variable x_v and its ℓ copies $\big($say $x_v^1, x_v^2, \ldots x_v^\ell\big)$, where $\ell := k \cdot (r - k + 1) + k \cdot q$. We force these copies to take the same truth value as x_v via the equations $x_v \oplus x_v^1 = 0, x_v^1 \oplus x_v^2 = 0, \ldots, x_v^{\ell-1} \oplus x_v^\ell = 0$. For every edge $e = uv \in E(G)$, we add variables y_e and z_e, and also the equation $x_u \oplus x_v \oplus y_e \oplus z_e = 1$.

For any $1 \leqslant i \leqslant k$, consider the i^{th} color-class, say $V_i = \{v_i^1, v_i^2, \ldots, v_i^N\}$. First, we add $N/3$ many *Stage 1 dummy variables* $\big($say $d_{i,1}^1$, $d_{i,1}^2$,…,$d_{i,1}^{N/3}\big)$, group the x variables corresponding to the vertices of V_i into $N/3$ triplets, and add $N/3$ equations that equate the xor of a triplet's variables and a dummy variable to 0. More precisely, we add the following $N/3$ equations:

$$\big(x_{v_i^1} \oplus x_{v_i^2} \oplus x_{v_i^3}\big) \oplus d_{i,1}^1 = 0,\ \big(x_{v_i^4} \oplus x_{v_i^5} \oplus x_{v_i^6}\big) \oplus d_{i,1}^2 = 0,\ \ldots,\ \big(x_{v_i^{N-2}} \oplus x_{v_i^{N-1}} \oplus x_{v_i^N}\big) \oplus d_{i,1}^{N/3} = 0$$

Next, we repeat the same process as follows: We introduce $N/3^2$ many *Stage 2 dummy variables* $\big($say $d_{i,2}^1$, $d_{i,2}^2$, …, $d_{i,2}^{N/3^2}\big)$, group the $N/3$ many Stage 1 dummy variables into $N/3^2$ triplets, and add $N/3^2$ equations that equate the xor of a triplet's Stage 1 dummy variables and a Stage 2 dummy variable to 0. More precisely, we add the following $N/3^2$ equations:

$$(d^1_{i,1} \oplus d^2_{i,1} \oplus d^3_{i,1}) \oplus d^1_{i,2} = 0,\ (d^4_{i,1} \oplus d^5_{i,1} \oplus d^6_{i,1}) \oplus d^2_{i,2} = 0,\ \ldots,\ (d^{N/3-2}_{i,1} \oplus d^{N/3-1}_{i,1} \oplus d^{N/3}_{i,1}) \oplus d^{N/3^2}_{i,2} = 0$$

Repeating the same procedure, we add the following $N/3^3$, $N/3^4$, …, $N/3^q = 2$ equations:

$$(d^1_{i,2} \oplus d^2_{i,2} \oplus d^3_{i,2}) \oplus d^1_{i,3} = 0,\ (d^4_{i,2} \oplus d^5_{i,2} \oplus d^6_{i,2}) \oplus d^2_{i,3} = 0,\ \ldots,\ (d^{N/3^2-2}_{i,2} \oplus d^{N/3^2-1}_{i,2} \oplus d^{N/3^2}_{i,2}) \oplus d^{N/3^3}_{i,3} = 0$$

$$(d^1_{i,3} \oplus d^2_{i,3} \oplus d^3_{i,3}) \oplus d^1_{i,4} = 0,\ (d^4_{i,3} \oplus d^5_{i,3} \oplus d^6_{i,3}) \oplus d^2_{i,4} = 0,\ \ldots,\ (d^{N/3^3-2}_{i,3} \oplus d^{N/3^3-1}_{i,3} \oplus d^{N/3^3}_{i,3}) \oplus d^{N/3^4}_{i,4} = 0$$

$$\vdots$$

$$\left(d^1_{i,q-1} \oplus d^2_{i,q-1} \oplus d^3_{i,q-1}\right) \oplus d^1_{i,q} = 0,\ \left(d^4_{i,q-1} \oplus d^5_{i,q-1} \oplus d^6_{i,q-1}\right) \oplus d^2_{i,q} = 0$$

Next, we add $B+1$ *auxiliary variables* (say $D^1_i, \ldots, D^{B+1}_i$) and the following equations:

$$\left(d^1_{i,q} \oplus d^2_{i,q}\right) \oplus D^1_i = 0,\ \left(d^1_{i,q} \oplus d^2_{i,q}\right) \oplus D^2_i = 0,\ \ldots,\ \left(d^1_{i,q} \oplus d^2_{i,q}\right) \oplus D^{B+1}_i = 0,$$

where $B := k \cdot (\ell+1) + k \cdot (r-k+1) + k \cdot q$ is the budget that we set on the total number of overlaps. That is, we set $d = n - B$, where n denotes the number of variables in ϕ. Now, we prove that (G, k) is a YES instance of Multicolored Clique if and only if (ϕ, d) is a YES instance of Max Differ Affine-SAT.

We first argue the forward direction. In the first assignment, we set i) all x and y variables to 0, ii) all z variables to 1, and iii) all dummy and auxiliary variables to 0. In the second assignment, we assign i) 0 to the k many x variables that correspond to the multi-colored clique's vertices, ii) 1 to the remaining x variables, iii) 0 to all z variables, iv) 0 to the $k \cdot (r-k+1)$ many y variables that correspond to those edges that have one endpoint inside the multi-colored clique and the other endpoint outside it, v) 1 to the remaining y variables, and vi) 1 to all auxiliary variables. Also, in the second assignment, for each $1 \leqslant i \leqslant k$, we assign i) 0 to that Stage 1 dummy variable which was grouped with the x variable corresponding to the multi-colored clique's vertex from the i^{th} color-class, 0 to that Stage 2 dummy variable which was grouped with this Stage 1 dummy variable, 0 to that Stage 3 dummy variable which was grouped with this Stage 2 dummy variable, and so on …, and ii) 1 to the remaining dummy variables. It can be verified that these two assignments satisfy ϕ, and they overlap on B many variables.

We argue the reverse direction of the equivalence. First, we show that each of the k color-classes has at least one vertex on whose corresponding x variable the two assignments overlap. Consider any $1 \leqslant i \leqslant k$. Since the $B+1$ auxiliary variables are forced to take the same truth value and there are only at most B overlaps, the two assignments must differ on them. This forces the two assignments to overlap on one of the two Stage q dummy variables. Further, this forces at least one overlap amongst the three Stage $q-1$ dummy variables that were grouped with this Stage q dummy variable. This effect propagates to lower-indexed stages, and eventually forces at least one overlap amongst the x variables corresponding to the vertices of the i^{th} color-class.

Next, we show that each of the k color-classes has at most one vertex on whose corresponding x variable the two assignments overlap. Suppose not. Then, there are at least two overlaps amongst the x variables corresponding to the vertices of some color class. Also, based on the previous paragraph, we know that there is at least one overlap amongst the x variables corresponding to the vertices of each of the remaining $k-1$ color classes. Therefore, overall, there are at least $k+1$ many overlaps amongst the x variables. So, the contribution of these x variables and their copies to the total number of overlaps becomes $\geqslant (k+1) \cdot (\ell+1) = B+1$. However, this exceeds the budget B on the number of overlaps, which is a contradiction.

Based on the previous two paragraphs, we know that for each $1 \leqslant i \leqslant k$, there is exactly one overlap amongst the x variables corresponding to the vertices of the i^{th} color class. Finally, we show that the set, say S, formed by these k vertices is the desired multi-colored clique. Suppose not. Then, there are $> k \cdot (r-k+1)$ edges that have one endpoint in S and the other endpoint outside S. Also, for each such edge, the two assignments must overlap on one of its corresponding y and z variables. Therefore, we have $> k \cdot (r-k+1)$ overlaps on the y and z variables. Also, $k \cdot q$ overlaps are forced on the dummy variables via the equations added in the grouping procedure. Thus, overall, the total number of overlaps exceeds B, which is a contradiction. This concludes a proof sketch of Theorem 2. ◀

If, on the other hand, all equations in the formula have at most two variables, then both problems turn out to be tractable. We describe this algorithm next.

▶ **Theorem 3.** *Both* Exact Differ Affine-SAT *and* Max Differ Affine-SAT *are polynomial-time solvable on 2-affine formulas.*

Proof. Consider an instance (ϕ, d) of Exact Differ Affine-SAT, where ϕ is a 2-affine formula. First, we construct a graph G_0 as follows: Introduce a vertex for every variable of ϕ. For every equation of the form $x \oplus y = 0$ in ϕ, add the edge xy. We compute the connected components of G_0. Observe that for each component C of G_0, the equations of ϕ corresponding to the edges of C are simultaneously satisfied if and only if all variables of C take the same truth value. So, any pair of satisfying assignments of ϕ either overlap on all variables in C, or differ on all variables in C. Thus, we replace all variables in C by a single variable, and set its weight to be the size of C. More precisely, i) we remove all but one variable (say z) of C from the variable-set of ϕ, ii) we remove all those equations from ϕ that correspond to the edges of C, iii) for every variable $v \in C \setminus \{z\}$, we replace the remaining appearances of v in ϕ (i.e., in equations of the form $v \oplus \_ = 1$) with z, and iv) we set the weight of z to be the number of variables in C. Let ϕ' denote the variable-weighted affine formula so obtained. Then, our goal is to decide whether ϕ' has a pair of satisfying assignments such that the weights of the variables at which they differ add up to exactly d.

Note that all equations in ϕ' are of the form $x \oplus y = 1$. Next, we construct a vertex-weighted graph G_1 as follows: Introduce a vertex for every variable of ϕ', and assign it the same weight as that of its corresponding variable. For every equation $x \oplus y = 1$ of ϕ', add the edge xy. We compute the connected components of G_1. Then, we run a bipartiteness-testing algorithm on each component of G_1. Suppose that there is a component C of G_1 that is not bipartite. Then, there is an odd-length cycle in C, say with vertices $x_1, x_2, \ldots, x_{2\ell}, x_{2\ell+1}$ (in that order). Note that the edges of this cycle correspond to the equations $x_1 \oplus x_2 = 1$, $x_2 \oplus x_3 = 1, \ldots, x_{2\ell} \oplus x_{2\ell+1} = 1$, $x_{2\ell+1} \oplus x_1 = 1$ in ϕ'. Adding (modulo 2) these $2\ell+1$ equations, we get LHS $= (2 \cdot x_1 + 2 \cdot x_2 + \ldots + 2 \cdot x_{2\ell+1}) \bmod 2 = 0$, and RHS $= (2\ell+1) \bmod 2 = 1$. So, these $2\ell+1$ equations of ϕ' cannot be simultaneously satisfied. Thus, we return NO. Now, assume that all components of G_1 are bipartite. See Figure 1 for an example.

Let $C_1, \ldots, C_p$ denote the connected components of G_1. Consider any $1 \leqslant i \leqslant p$. Let A and B denote the parts of the bipartite component C_i. Observe that the equations of ϕ' corresponding to the edges of C_i are simultaneously satisfied if and only if either i) all variables in A are set to 1, and all variables in B are set to 0, or ii) all variables in A are set to 0, and all variables in B are set to 1. So, any pair of satisfying assignments of ϕ' either overlap on all variables in C_i, or differ on all variables in C_i. Thus, our problem amounts to deciding whether there is a subset of components of G_1 whose collective weight is exactly d. That is, our goal is to decide whether there exists $X \subseteq [p]$ such that $\sum_{i \in X} \mathtt{weight}(C_i) = d$,

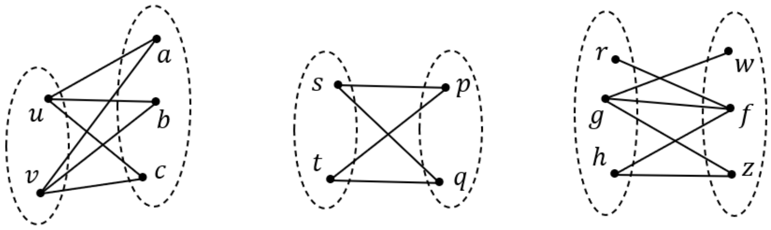


Figure 1 This figure shows the bipartite components of the graph G_1 constructed in the proof of Theorem 3, when the 2-affine formula ϕ' consists of the following equations: $u \oplus a = 1$, $u \oplus b = 1$, $u \oplus c = 1$, $v \oplus a = 1$, $v \oplus b = 1$, $v \oplus c = 1$, $s \oplus p = 1$, $s \oplus q = 1$, $t \oplus p = 1$, $t \oplus q = 1$, $r \oplus f = 1$, $g \oplus w = 1$, $g \oplus f = 1$, $g \oplus z = 1$, $h \oplus f = 1$, $h \oplus z = 1$.

where $weight(C_i)$ denotes the sum of the weights of the variables in C_i. To do so, we use the algorithm for Subset Sum problem with $\{weight(C_1), \ldots, weight(C_p)\}$ as the multi-set of integers and d as the target sum. This proves Theorem 3. The algorithm described here works almost as it is for Max Differ Affine-SAT too. In the last step, instead of reducing to Subset Sum problem, we simply check whether the collective weight of all components of G_1 is at least d. That is, if $\sum_{i=1}^{p} weight(C_i) \geqslant d$, we return YES; otherwise, we return NO. Thus, Max Differ Affine-SAT is polynomial-time solvable on 2-affine formulas. ◀

We now turn to the parameterized complexity of Exact Differ Affine-SAT and Max Differ Affine-SAT when parameterized by the number of variables that differ in the two solutions. It turns out that the exact version of the problem is W[1]-hard, while the maximization question is FPT. We first show the hardness of Exact Differ Affine-SAT by a reduction from Exact Even Set.

▶ **Theorem 4.** *Exact Differ Affine-SAT is* W[1]*-hard in the parameter* d.

Proof. We describe a reduction from Exact Even Set. Consider an instance $(\mathcal{U}, \mathcal{F}, k)$ of Exact Even Set. We construct an affine formula ϕ as follows: For every element u in the universe $\mathcal{U}$, introduce a variable x_u. For every set S in the family $\mathcal{F}$, introduce the equation $\bigoplus_{u \in S} x_u = 0$. We set $d = k$. We prove that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Even Set if and only if (ϕ, d) is a YES instance of Exact Differ Affine-SAT. At a high level, we argue this equivalence as follows: In the forward direction, we show that the two desired satisfying assignments are i) the all 0 assignment, and ii) the assignment that assigns 1 to the variables that correspond to the elements of the given even set, and assigns 0 to the remaining variables. In the reverse direction, we show that the desired even set consists of those elements of the universe that correspond to the variables on which the two given satisfying assignments differ. We now argue the equivalence.

Forward direction. Suppose that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Even Set. That is, there is a set $X \subseteq \mathcal{U}$ of size exactly k such that $|X \cap S|$ is even for all sets S in the family $\mathcal{F}$. Let σ_1 and σ_2 be assignments of ϕ defined as follows: For every $u \in X$, σ_1 sets x_u to 0, and σ_2 sets x_u to 1. For every $u \in \mathcal{U} \setminus X$, both σ_1 and σ_2 set x_u to 0. Note that σ_1 and σ_2 differ on exactly $|X| = k$ variables. Consider any set S in the family $\mathcal{F}$. The equation corresponding to S in the formula ϕ is $\bigoplus_{u \in S} x_u = 0$. All variables in the left-hand side are set to 0 by σ_1. Also, the number of variables in the left-hand side that are set to 1 by σ_2 is $|X \cap S|$, which is an even number. Therefore, the left-hand side evaluates to 0 under both σ_1 and σ_2. So, σ_1 and σ_2 are satisfying assignments of ϕ. Hence, (ϕ, k) is a YES instance of Exact Differ Affine-SAT.

Reverse direction. Suppose that (ϕ, k) is a YES instance of Exact Differ Affine-SAT. That is, there are satisfying assignments σ_1 and σ_2 of ϕ that differ on exactly k variables. Let X denote the k-sized set $\{u \in \mathcal{U} \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } x_u\}$. Consider any set S in the family $\mathcal{F}$. The equation corresponding to S in the formula ϕ is $\bigoplus_{u \in S} x_u = 0$. We split the left-hand side into two parts to express this equation as $\underbrace{\bigoplus_{u \in S \setminus X} x_u}_{A} \oplus \underbrace{\bigoplus_{u \in X \cap S} x_u}_{B} = 0$. Note that σ_1 and σ_2 overlap on all variables in the first part, i.e., A. So, A evaluates to the same truth value under both assignments. Thus, as both σ_1 and σ_2 satisfy this equation, they must assign the same truth value to the second part, i.e., B, as well. Also, σ_1 and σ_2 differ on all variables in B. So, for its truth value to be same under both assignments, B must have an even number of variables. That is, $|X \cap S|$ must be even. Hence, $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Even Set.

This proves Theorem 4. ◀

We now turn to the FPT algorithm for Max Differ Affine-SAT, which is based on obtaining solutions using Gaussian elimination and working with the free variables: if the set of free variables F is "large", we can simply set them differently and force the dependent variables, and guarantee ourselves a distinction on at least $|F|$ variables. Note that this is the step that would not work as-is for the exact version of the problem. If the number of free variables is bounded, we can proceed by guessing the subset of free variables on which the two assignments differ. We make these ideas precise in the proof of Theorem 5. Also, in the proof of Theorem 6, we show that Max Differ Affine-SAT has a polynomial kernel in the parameter d.

▶ **Theorem 5.** *Max Differ Affine-SAT admits an algorithm with running time* $\mathcal{O}^\star(2^d)$.

Proof. Consider an instance (ϕ, d) of Max Differ Affine-SAT. We use Gaussian elimination to find the solution set of ϕ in polynomial-time. If ϕ has no solution, we return NO. If ϕ has a unique solution and $d = 0$, we return YES. If ϕ has a unique solution and $d \geqslant 1$, we return NO. Now, assume that ϕ has multiple solutions. Let F denote the set of all free variables. Suppose that $|F| \geqslant d$. Let σ_1 denote the solution of ϕ obtained by setting all free variables to 0, and then setting the forced variables to take values as per their dependence on the free variables. Similarly, let σ_2 denote the solution of ϕ obtained by setting all free variables to 1, and then setting the forced variables to take values as per their dependence on the free variables. Note that σ_1 and σ_2 differ on all free variables (and possibly some forced variables too). So, overall, they differ on at least $|F| \geqslant d$ variables. Thus, we return YES. Now, assume that $|F| \leqslant d - 1$. We guess the subset $D \subseteq F$ of free variables on which two desired solutions (say σ_1 and σ_2) differ. Note that there are $2^{|F|} \leqslant 2^{d-1}$ such guesses.

First, consider any forced variable x that depends on an odd number of free variables from D. That is, the expression for its value is the XOR of an odd number of free variables from D (possibly along with the constant 1 and/or some free variables from $F \setminus D$). Then, note that this expression takes different truth values under σ_1 and σ_2. That is, σ_1 and σ_2 differ on x. Next, consider any forced variable x that depends on an even number of free variables from D. That is, the expression for its value is the XOR of an even number of free variables from D (possibly along with the constant 1 and/or some free variables from $F \setminus D$). Then, note that this expression takes the same truth value under σ_1 and σ_2. That is, σ_1 and σ_2 overlap on x. Thus, overall, these two solutions differ on i) all free variables from D, and ii) all those forced variables that depend upon an odd number of free variables from D. If the total count of such variables is $\geqslant d$ for some guess D, we return YES. Otherwise, we return NO. This concludes the proof. ◀

▶ **Theorem 6.** *Max Differ Affine-SAT admits a kernel with $\mathcal{O}(d^2)$ variables and $\mathcal{O}(d^2)$ equations.*

Proof. Consider an instance (ϕ, d) of Max Differ Affine-SAT. We use Gaussian elimination to find the solution set of ϕ in polynomial-time. Then, as in the proof of Theorem 5, i) we return NO if ϕ has no solution, or if ϕ has a unique solution and $d \geqslant 1$, ii) we return YES if ϕ has a unique solution and $d = 0$, or if ϕ has multiple solutions with at least d free variables. Now, assume that ϕ has multiple solutions with at most $d - 1$ free variables. Note that the system of linear equations formed by the expressions for the values of forced variables is an affine formula (say ϕ') that is equivalent to ϕ. That is, ϕ' and ϕ have the same solution sets. So, we work with the instance (ϕ', d) in the remaining proof.

Suppose that there is a free variable, say x, such that at least $d - 1$ forced variables depend on x. That is, there are at least $d - 1$ forced variables such that the expressions for their values are the XOR of x (possibly along with the constant 1 and/or some other free variables). Let σ_1 denote the solution of ϕ' obtained by setting all free variables to 0, and then setting the forced variables to take values as per their dependence on the free variables. Let σ_2 denote the solution of ϕ' obtained by setting x to 1 and the remaining free variables to 0, and then setting the forced variables to take values as per their dependence on the free variables. Note that σ_1 and σ_2 differ on x, and also on each of the $\geqslant d - 1$ forced variables that depend on x. So, overall, σ_1 and σ_2 differ on at least d variables. Thus, we return YES.

Now, assume that for every free variable x, there are at most $d - 2$ forced variables that depend on x. So, as there are at most $d - 1$ free variables, it follows that there are at most $(d - 1) \cdot (d - 2)$ forced variables that depend on at least one free variable. The remaining forced variables are the ones that do not depend on any free variable. That is, any such forced variable y is set to a constant (i.e., 0 or 1) as per the expression for its value. We remove the variable y and its corresponding equation (i.e., $y = 0$ or $y = 1$) from ϕ', and we leave d unchanged. This is safe because y takes the same truth value under all solutions of ϕ'. Note that the affine formula so obtained has at most $d - 1$ free variables and at most $(d - 1) \cdot (d - 2)$ forced variables. So, overall, it has at most $(d - 1)^2$ variables. Also, it has at most $(d - 1) \cdot (d - 2)$ equations. This concludes the proof. ◀

We finally turn to the "dual" parameter, $n - d$: the number of variables on which the two assignments sought *overlap*. We show that both the exact and maximization variants for affine formulas are W[1]-hard in this parameter by reductions from Exact Odd Set and Odd Set, respectively.

▶ **Theorem 7.** *The problems Exact Differ Affine-SAT and Max Differ Affine-SAT are W[1]-hard in the parameter $n - d$.*

Proof. We describe a reduction from Exact Odd Set. Consider an instance $(\mathcal{U}, \mathcal{F}, k)$ of Exact Odd Set. We construct an affine formula ϕ as follows: For every element u in the universe $\mathcal{U}$, introduce a variable x_u. For every odd-sized set S in the family $\mathcal{F}$, introduce the equation $\bigoplus_{u \in S} x_u = 1$. For every even-sized set S in the family $\mathcal{F}$, introduce $k+1$ variables $y_S, z_S^1, z_S^2, \ldots, z_S^k$, and the equations $y_S \oplus z_S^1 = 0, y_S \oplus z_S^2 = 0, \ldots, y_S \oplus z_S^k = 0$ and $\bigoplus_{u \in S} x_u \oplus y_S = 0$. The number of variables in ϕ is $n = |\mathcal{U}| + (k+1) \cdot \left|\left\{S \in \mathcal{F} \mid |S| \text{ is even}\right\}\right|$. We set $d = n - k$. We prove that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Odd Set if and only if (ϕ, d) is a YES instance of Exact Differ Affine-SAT. At a high level, we argue this equivalence as follows: In the forward direction, we show that the two desired satisfying assignments are i) the assignment that sets all y and z variables to 0 and all x variables to 1, and ii) the assignment that sets all y and z variables to 1, assigns 1 to all those x variables that correspond to the elements of the given odd set, and assigns 0 to the remaining x variables. In the reverse direction, we show that the two assignments must differ on all y and z variables (and so, all k overlaps are restricted to occur at x variables), and the desired odd set consists of those elements of the universe that correspond to the x variables on which the two assignments overlap. We present a full proof of this equivalence in Theorem 7. This reduction also works with Odd Set as the source problem and Max Differ Affine-SAT as the target problem. So, Max Differ Affine-SAT is also W[1]-hard in the parameter $n - d$.

Forward direction. Suppose that $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Differ Affine-SAT. That is, there is a set $X \subseteq \mathcal{U}$ of size exactly k such that $|X \cap S|$ is odd for all sets S in the family $\mathcal{F}$. Let σ_1 and σ_2 be assignments of ϕ defined as follows: For every even-sized set S in the family $\mathcal{F}$, σ_1 sets $y_S, z_S^1, z_S^2, \ldots, z_S^k$ to 0, and σ_2 sets $y_S, z_S^1, z_S^2, \ldots, z_S^k$ to 1. For every $u \in X$, both σ_1 and σ_2 set x_u to 1. For every $u \in \mathcal{U} \setminus X$, σ_1 sets x_u to 1, and σ_2 sets x_u to 0. Note that σ_1 and σ_2 overlap on exactly $|X| = k$ variables (and so, they differ on exactly $n - k$ variables). Now, we show that σ_1 and σ_2 are satisfying assignments of ϕ.

First, we argue that σ_1 and σ_2 satisfy the equations of ϕ that were added corresponding to odd-sized sets of the family $\mathcal{F}$. Consider any odd-sized set S in the family $\mathcal{F}$. The equation corresponding to S in the formula ϕ is $\bigoplus\limits_{u \in S} x_u = 1$. The number of variables in the left-hand side that are set to 1 by σ_2 is $|X \cap S|$, which is an odd number. Also, all $|S|$ (again, which is an odd number) variables in the left-hand side are set to 1 by σ_1. Therefore, the left-hand side evaluates to 1 under both σ_1 and σ_2. So, both these assignments satisfy the equation $\bigoplus\limits_{u \in S} x_u = 1$.

Next, we argue that σ_1 and σ_2 satisfy the equations of ϕ that were added corresponding to even-sized sets of the family $\mathcal{F}$. Consider any even-sized set S in the family $\mathcal{F}$. The $k+1$ equations corresponding to S in the formula ϕ are $y_S \oplus z_S^1 = 0, y_S \oplus z_S^2 = 0, \ldots, y_S \oplus z_S^k = 0$ and $\bigoplus\limits_{u \in S} x_u \oplus y_S = 0$. Consider any of the first k equations, say $y_S \oplus z_S^i = 0$, where $1 \leqslant i \leqslant k$. Both variables on the left-hand side, i.e., y_S and z_S^i, are assigned the same truth value, i.e., both 0 by σ_1 and both 1 by σ_2. So, both these assignments satisfy the equation $y_S \oplus z_S^i = 0$. Next, consider the last equation, i.e., $\bigoplus\limits_{u \in S} x_u \oplus y_S = 0$. The number of variables amongst $x_u\big|_{u \in S}$ that are set to 1 by σ_2 is $|X \cap S|$, which is an odd number. Also, the variable y_S is set to 1 by σ_2. Therefore, overall, the number of variables in the left-hand side that are set to 1 by σ_2 is even. Also, σ_1 sets all variables on the left-hand side to 1 except y_S. That is, it sets all the $|S|$ (again, which is an even number) variables $x_u\big|_{u \in S}$ to 1. Therefore, the left-hand side evaluates to 0 under both σ_1 and σ_2. So, both these assignments satisfy the equation $\bigoplus\limits_{u \in S} x_u \oplus y_S = 0$.

Hence, $(\phi, n - k)$ is a YES instance of Exact Differ Affine-SAT.

Reverse direction. Suppose that $(\phi, n - k)$ is a YES instance of Exact Differ Affine-SAT. That is, there are satisfying assignments σ_1 and σ_2 of ϕ that overlap on exactly k variables. Consider any even-sized set S in the family $\mathcal{F}$. As σ_1 satisfies the equations $y_S \oplus z_S^1 = 0, y_S \oplus z_S^2 = 0, \ldots, y_S \oplus z_S^k = 0$, it must assign the same truth value to all the $k+1$ variables $y_S, z_S^1, z_S^2, \ldots, z_S^k$. Similarly, σ_2 must assign the same truth value to $y_S, z_S^1, z_S^2, \ldots, z_S^k$. Therefore, either σ_1 and σ_2 overlap on all these $k+1$ variables, or they differ on all these $k+1$ variables. So, as there are only k overlaps, σ_1 and σ_2 must differ on $y_S, z_S^1, z_S^2, \ldots, z_S^k$. Thus, all the k overlaps occur at x variables. Let X denote the k-sized set $\{u \in \mathcal{U} \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } x_u\}$. Now, we show that $|X \cap S|$ is odd for all sets S in $\mathcal{F}$.

First, we argue that X has odd-sized intersection with all odd-sized sets of the family $\mathcal{F}$. Consider any odd-sized set S in the family $\mathcal{F}$. The equation corresponding to S in the formula ϕ is $\bigoplus_{u \in S} x_u = 1$. We split the left-hand side into two parts to express this equation as $\underbrace{\bigoplus_{u \in X \cap S} x_u}_{A} \oplus \underbrace{\bigoplus_{u \in S \setminus X} x_u}_{B} = 1$. Note that σ_1 and σ_2 overlap on all variables in the first part, i.e., A. So, A evaluates to the same truth value under both assignments. Thus, as both σ_1 and σ_2 satisfy this equation, they must assign the same truth value to the second part, i.e., B, as well. Also, σ_1 and σ_2 differ on all variables in B. So, for its truth value to be same under both assignments, B must have an even number of variables. That is, $|S \setminus X|$ must be even. Now, as $|S|$ is odd and $|S \setminus X|$ is even, we infer that $|X \cap S| = |S| - |S \setminus X|$ is odd.

Next, we argue that X has odd-sized intersection with all even-sized sets of the family $\mathcal{F}$. Consider any even-sized set S in the family $\mathcal{F}$. Amongst the $k+1$ equations corresponding to S in the formula ϕ, consider the last equation, i.e., $\bigoplus_{u \in S} x_u \oplus y_S = 0$. We split the left-hand side into two parts to express this equation as $\underbrace{\bigoplus_{u \in X \cap S} x_u}_{A} \oplus \underbrace{\bigoplus_{u \in S \setminus X} x_u \oplus y_S}_{B} = 0$. Note that σ_1 and σ_2 overlap on all variables in the first part, i.e., A. So, A evaluates to the same truth value under both assignments. Thus, as both σ_1 and σ_2 satisfy this equation, they must assign the same truth value to the second part, i.e., B. Also, σ_1 and σ_2 differ on all variables in B. So, for its truth value to be same under both assignments, B must have an even number of variables. That is, $|S \setminus X| + 1$ must be even. Now, as $|S|$ is even and $|S \setminus X|$ is odd, we infer that $|X \cap S| = |S| - |S \setminus X|$ is odd. Hence, $(\mathcal{U}, \mathcal{F}, k)$ is a YES instance of Exact Odd Set. ◀

4 2-CNF formulas

In this section, we explore the classical and parameterized complexity of Max Differ 2-SAT and Exact Differ 2-SAT. We first show that these problems are polynomial time solvable on $(2,2)$-CNF formulas by constructing a graph corresponding to the instance and observing some structural properties of that graph. Then we show that both of these problems are W[1]-hard with respect to the parameter d. We begin by proving the following theorem.

▶ **Theorem 8.** *Max Differ 2-SAT is polynomial-time solvable on* $(2,2)$*-CNF formulas.*

We use similar ideas in the proof of Theorem 8 to show that Exact Differ 2-SAT can also be solved in polynomial time on $(2,2)$-CNF formulas. This requires more careful analysis of the graph constructed and a reduction to Subset Sum problem, as we want the individual contributions, in terms of number of variables where the assignments differ, to sum up to an exact value. We show the result in the following theorem.

▶ **Theorem 9.** *Exact Differ 2-SAT is polynomial-time solvable on* $(2,2)$*-CNF formulas.*

Due to lack of space, the proofs of these results are deferred to a full version of the paper. Looking at the parameterized complexity of Exact Differ 2-SAT and Max Differ 2-SAT with respect to the parameter d, we establish the following hardness result.

▶ **Theorem 10.** *Exact/Max Differ 2-SAT is* W[1]*-hard in the parameter* d.

We describe a reduction from Independent Set. Consider an instance (G, k) of Independent Set. We construct a 2-CNF formula ϕ as follows: For every vertex $v \in V(G)$, introduce two variables x_v and y_v; we refer to them as x-variable and y-variable respectively. For every edge $uv \in E(G)$, i) we add a clause that consists of the x-variables corresponding to the vertices u and v, i.e., $x_u \vee x_v$, and ii) we add a clause that consists of the y-variables corresponding to the vertices u and v, i.e., $y_u \vee y_v$. For every pair of vertices $u, v \in V(G)$, we add a clause that consists of the x-variable corresponding to u and the y-variable corresponding to v, i.e., $x_u \vee y_v$. We set $d = 2k$. We prove that (G, k) is a YES instance of Independent Set if and only if (ϕ, d) is a YES instance of Exact Differ 2-SAT.

At a high level, we argue this equivalence as follows: In the forward direction, we show that the two desired satisfying assignments are i) the assignment that assigns 0 to all x-variables corresponding to the vertices of the given independent set, and 1 to the remaining variables, and ii) the assignment that assigns 0 to all y-variables corresponding to the vertices of the given independent set, and 1 to the remaining variables. In the reverse direction, we partition the set of variables on which the two given assignments differ into two parts: i) one part consists of those variables that are set to 1 by the first assignment, and 0 by the second assignment, and ii) the other part consists of those variables that are set to 0 by the first assignment, and 1 by the second assignment. Then, we show that at least one of these two parts has the desired size, and it is not a mix of x-variables and y-variables. That is, either it has only x-variables, or it has only y-variables. Finally, we show that the vertices that correspond to the variables in this part form the desired independent set. A detailed proof of equivalence is deferred to a full version of the paper.

5 Hitting formulas

In this section, we consider hitting formulas, and we show that both its diverse variants, i.e., Exact Differ Hitting-SAT and Max Differ Hitting-SAT, are polynomial-time solvable.

▶ **Theorem 11.** *Exact Differ Hitting-SAT admits a polynomial-time algorithm.*

Proof. Consider an instance (ϕ, d) of Exact Differ Hitting-SAT, where ϕ is a hitting formula with m clauses (say $C_1, \dots, C_m$) on n variables. For every $1 \leqslant i \leqslant m$, let $vars(C_i)$ denote the set of all variables that appear in the clause C_i. For every $1 \leqslant i, j \leqslant m$, let $\lambda(i, j)$ denote the number of variables $x \in vars(C_i) \cap vars(C_j)$ such that x appears as a positive literal in one clause, and as a negative literal in the other clause. Note that

$$\begin{aligned}
&\big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables, and both } \sigma_1 \text{ and } \sigma_2 \text{ satisfy } \phi\}\big| \\
&= \big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables}\}\big| \\
&\quad - \big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables, and } \sigma_1 \text{ falsifies } \phi\}\big| \\
&\quad - \big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables, and } \sigma_2 \text{ falsifies } \phi\}\big| \\
&\quad + \big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables, and both } \sigma_1 \text{ and } \sigma_2 \text{ falsify } \phi\}\big| \\
&= 2^n \cdot \binom{n}{d} - \big|\{\sigma_1 \mid \sigma_1 \text{ falsifies } \phi\}\big| \cdot \binom{n}{d} - \big|\{\sigma_2 \mid \sigma_2 \text{ falsifies } \phi\}\big| \cdot \binom{n}{d} \\
&\quad + \sum_{i=1}^{m} \sum_{j=1}^{m} \underbrace{\big|\{(\sigma_1, \sigma_2) \mid \sigma_1 \text{ and } \sigma_2 \text{ differ on } d \text{ variables, } \sigma_1 \text{ falsifies } C_i, \text{ and } \sigma_2 \text{ falsifies } C_j\}\big|}_{\alpha(i,j)} \\
&= \left(2^n - 2 \cdot \sum_{i=1}^{m} 2^{n - |vars(C_i)|}\right) \cdot \binom{n}{d} + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha(i, j)
\end{aligned}$$

Consider any $1 \leqslant i, j \leqslant m$. Let us find an expression for $\alpha(i, j)$. That is, let us count the number of pairs (σ_1, σ_2) of assignments of ϕ such that σ_1 and σ_2 differ on d variables, σ_1 falsifies C_i, and σ_2 falsifies C_j. Since σ_1 falsifies C_i, it must set every variable in $vars(C_i)$

such that its corresponding literal in the clause C_i is falsified. That is, for every $x \in vars(C_i)$, if x appears as a positive literal in C_i, then σ_1 must set x to 0; otherwise, it must set x to 1. Similarly, since σ_2 falsifies C_j, it must set every variable in $vars(C_j)$ such that its corresponding literal in the clause C_j is falsified.

There is just one choice for the truth values assigned to the variables in $vars(C_i) \cap vars(C_j)$ by σ_1 and σ_2. Also, note that for every variable x in $vars(C_i) \cap vars(C_j)$, if x appears as a positive literal in one clause and as a negative literal in the other clause, then σ_1 and σ_2 differ on x; otherwise, they overlap on x. So, overall, σ_1 and σ_2 differ on $\lambda(i, j)$ variables amongst the variables in $vars(C_i) \cap vars(C_j)$.

We go over all possible choices for the numbers of variables on which σ_1 and σ_2 differ (say d_1, d_2 and d_3 many variables) amongst the variables in $vars(C_i) \setminus vars(C_j), vars(C_j) \setminus vars(C_i)$ and $vars(\phi) \setminus (vars(C_i) \cup vars(C_j))$ respectively, where $vars(\phi)$ denotes the set of all variables of ϕ. As σ_1 and σ_2 differ on d variables in total, we have $\lambda(i, j)+d_1+d_2+d_3 = d$.

There is just one choice for the truth values assigned to the variables in $vars(C_i)\setminus vars(C_j)$ by σ_1, and there are $\binom{|vars(C_i)\setminus vars(C_j)|}{d_1}$ choices for the truth values assigned to the variables in $vars(C_i) \setminus vars(C_j)$ by σ_2. Similarly, there is just one choice for the truth values assigned to the variables in $vars(C_j) \setminus vars(C_i)$ by σ_2, and there are $\binom{|vars(C_j)\setminus vars(C_i)|}{d_2}$ choices for the truth values assigned to the variables in $vars(C_j) \setminus vars(C_i)$ by σ_1.

There are $\binom{n-|vars(C_i)\cup vars(C_j)|}{d_3}$ choices for the d_3 variables on which σ_1 and σ_2 differ amongst the variables in $vars(\phi) \setminus (vars(C_i) \cup vars(C_j))$. For each variable x amongst these d_3 variables, there are two ways in which σ_1 and σ_2 can assign truth values to x. That is, either i) σ_1 sets x to 0 and σ_2 sets x to 1, or ii) σ_1 sets x to 1 and σ_2 sets x to 0. For each variable x amongst the remaining $n - |vars(C_i) \cup vars(C_j)| - d_3$ variables, there are again two ways in which σ_1 and σ_2 can assign truth values to x. That is, either i) both σ_1 and σ_2 set x to 1, or ii) both σ_1 and σ_2 set x to 0. So, overall, the number of ways in which σ_1 and σ_2 can assign truth values to the variables in $vars(\phi) \setminus (vars(C_i) \cup vars(C_j))$ is $\binom{n-|vars(C_i)\cup vars(C_j)|}{d_3} \cdot 2^{d_3} \cdot 2^{n-|vars(C_i)\cup vars(C_j)|-d_3}$.

Thus, we get the following expression for $\alpha(i, j)$:

$$2^{n-|vars(C_i)\cup vars(C_j)|} \cdot \sum_{\substack{d_1,d_2,d_3\geqslant 0: \\ d_1+d_2+d_3=d-\lambda(i,j)}} \binom{|vars(C_i)\setminus vars(C_j)|}{d_1}\binom{|vars(C_j)\setminus vars(C_i)|}{d_2}\binom{n-|vars(C_i)\cup vars(C_j)|}{d_3}$$

Plugging this into the previously obtained equality, we get an expression to count the number of pairs (σ_1, σ_2) of satisfying assignments of ϕ that differ on d variables. This expression can be evaluated in polynomial-time. If the count so obtained is non-zero, we return YES; otherwise, we return NO. This proves Theorem 11. Note that (ϕ, d) is a YES instance of Max Differ Hitting-SAT if and only if at least one of $(\phi, d), (\phi, d + 1), \dots, (\phi, n)$ is a YES instance of Exact Differ Hitting-SAT. Thus, as Exact Differ Hitting-SAT is polynomial-time solvable, so is Max Differ Hitting-SAT. ◀

6 Concluding remarks

In this work, we undertook a complexity-theoretic study of the problem of finding a diverse pair of satisfying assignments of a given Boolean formula, when restricted to affine, 2-CNF and hitting formulas. This problem can also be studied for i) other classes of formulas on which SAT is polynomial-time solvable, ii) more than two solutions, and iii) other notions of distance between assignments. An immediate open problem is to resolve the parameterized complexities of Exact Differ 2-SAT and Max Differ 2-SAT in the parameter $n - d$.

References

1 Ola Angelsmark and Johan Thapper. Algorithms for the maximum Hamming distance problem. In *International Workshop on Constraint Solving and Constraint Logic Programming*, pages 128–141. Springer, 2004. doi:10.1007/11402763_10.

2 Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information processing letters*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

3 Julien Baste, Michael R Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022. doi:10.1016/j.artint.2021.103644.

4 Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019. doi:10.3390/a12120254.

5 Endre Boros, Yves Crama, and Peter L Hammer. Polynomial-time inference of all valid implications for horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990. doi:10.1007/BF01531068.

6 Endre Boros, Peter L Hammer, and Xiaorong Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, 55(1):1–13, 1994. doi:10.1016/0166-218X(94)90033-7.

7 Michele Conforti, Gérard Cornuéjols, Ajai Kapoor, Kristina Vušković, and MR Rao. Balanced matrices. *Mathematical Programming: State of the Art*, pages 1–33, 1994.

8 Stephen A Cook. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 143–152. Association for Computing Machinery, 2023. doi:10.1145/3588287.3588297.

9 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Vilhelm Dahllöf. Algorithms for max Hamming exact satisfiability. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC)*, pages 829–838. Springer, 2005. doi:10.1007/11602613_83.

11 Mark de Berg, Andrés López Martínez, and Frits C. R. Spieksma. Finding diverse minimum s-t cuts. In *Proceedings of the 34th International Symposium on Algorithms and Computation ISAAC*, volume 283 of *LIPIcs*, pages 24:1–24:17, 2023. doi:10.4230/LIPIcs.ISAAC.2023.24.

12 William F Dowling and Jean H Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984. doi:10.1016/0743-1066(84)90014-1.

13 Rod G Downey, Michael R Fellows, Alexander Vardy, and Geoff Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM Journal on Computing*, 29(2):545–570, 1999. doi:10.1137/S0097539797323571.

14 Rodney G Downey and Michael R Fellows. Fixed-parameter intractability. In *1992 Seventh Annual Structure in Complexity Theory Conference*, pages 36–37. IEEE Computer Society, 1992.

15 Rodney G Downey, Michael R Fellows, et al. *Fundamentals of Parameterized Complexity*, volume 4. Springer, 2013.

16 S. Even, A. Shamir, and A. Itai. On the complexity of time table and multi-commodity flow problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 184–193. IEEE Computer Society, 1975. doi:10.1109/SFCS.1975.21.

17 Fedor V Fomin, Petr A Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

18 Fedor V Fomin, Petr A Golovach, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Diverse collections in matroids and graphs. *Mathematical Programming*, pages 1–33, 2023.

19 John Franco and Allen Van Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Applied Mathematics*, 125(2-3):177–214, 2003. doi:10.1016/S0166-218X(01)00358-4.

20 Aadityan Ganesh, HV Vishwa Prakash, Prajakta Nimbhorkar, and Geevarghese Philip. Disjoint stable matchings in linear time. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 94–105. Springer, 2021. doi:10.1007/978-3-030-86838-3_7.

21 Joseph F Grcar. How ordinary elimination became Gaussian elimination. *Historia Mathematica*, 38(2):163–218, 2011.

22 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3778–3786, 2021. doi:10.1609/aaai.v35i5.16495.

23 Gordon Hoi, Sanjay Jain, and Frank Stephan. A fast exponential time algorithm for max hamming distance x3sat. *arXiv preprint*, 2019. arXiv:1910.01293.

24 Gordon Hoi and Frank Stephan. Measure and conquer for max Hamming distance XSAT. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

25 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences (JCSS)*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

26 Kazuo Iwama. CNF-satisfiability test by counting and polynomial average time. *SIAM Journal on Computing*, 18(2):385–391, 1989. doi:10.1137/0218026.

27 Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.

28 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms (TALG)*, 15(3):1–20, 2019. doi:10.1145/3329863.

29 Melven R Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

30 Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

31 Harry R Lewis. Renaming a set of clauses as a Horn set. *Journal of the ACM (JACM)*, 25(1):134–135, 1978. doi:10.1145/322047.322059.

32 Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 102–115. Springer, 2006. doi:10.1007/11814948_13.

33 Neeldhara Misra, Harshil Mittal, and Saraswati Nanoti. Diverse non crossing matchings. In *CCCG*, pages 249–256, 2022.

34 Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory (JCT), Series B*, 82(1):102–117, 2001. doi:10.1006/jctb.2000.2026.

35 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM Symposium on Theory of Computing*, pages 216–226, 1978. doi:10.1145/800133.804350.

36 Maria Grazia Scutella. A note on Dowling and Gallier's top-down algorithm for propositional horn satisfiability. *The Journal of Logic Programming*, 8(3):265–273, 1990. doi:10.1016/0743-1066(90)90026-2.

37 Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015. doi:10.1109/JPROC.2015.2455034.

Easier Ways to Prove Counting Hard: A Dichotomy for Generalized #SAT, Applied to Constraint Graphs

MIT Hardness Group[1]
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Josh Brunner
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Jenny Diomidova
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Timothy Gomez
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Markus Hecher
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Frederick Stock
Miner School of Computer & Information Sciences, University of Massachusetts, Lowell, MA, USA

Zixiang Zhou
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Abstract

To prove #P-hardness, a single-call reduction from #2SAT needs a clause gadget to have exactly the same number of solutions for all satisfying assignments – no matter how many and which literals satisfy the clause. In this paper, we relax this condition, making it easier to find #P-hardness reductions. Specifically, we introduce a framework called Generalized #SAT where each clause contributes a term to the total count of solutions based on a given function of the literals. For two-variable clauses (a natural generalization of #2SAT), we prove a dichotomy theorem characterizing when Generalized #SAT is in FP versus #P-complete.

Equipped with these tools, we analyze the complexity of counting solutions to Constraint Graph Satisfiability (CGS), a framework previously used to prove NP-hardness (and PSPACE-hardness) of many puzzles and games. We prove CGS ASP-hard, meaning that there is a parsimonious reduction (with algorithmic bijection on solutions) from every NP search problem, which implies #P-completeness. Then we analyze CGS restricted to various subsets of features (vertex and edge types), and prove most of them either easy (in FP) or hard (#P-complete). Most of our results also apply to planar constraint graphs. CGS is thus a second powerful framework for proving problems #P-hard, with reductions requiring very few gadgets.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Complexity theory and logic

Keywords and phrases Counting, Computational Complexity, Sharp-P, Dichotomy, Constraint Graph Satisfiability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.51

Funding *Markus Hecher*: The author was funded by the Austrian Science Fund (FWF), grants J4656, the Society for Research Funding in Lower Austria (GFF NOE) grant ExzF-0004, as well as the Vienna Science and Technology Fund (WWTF) grant ICT19-065. Part of the research was carried out while visiting Simons Inst./UC Berkeley.

[1] Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT Hardness Group" (without "et al.").

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 51; pp. 51:1–51:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Counting solutions to NP search problems (i.e., problems in the complexity class #P) is an algorithmic analog of the field of combinatorics. Counting (#P) is also provably harder than deciding (NP): by a consequence of Toda's Theorem [10], any problem in the polynomial hierarchy has a deterministic polynomial-time reduction to a single counting problem in #P. Already from the foundational work by Valiant in the late 1970s [13, 14], we have problems where decision is in P yet counting is #P-hard. One notable example is 2SAT, where finding one solution is easy, but counting the solutions (#2SAT) is #P-hard.

How do we prove a problem #P-hard? Ideally, we would find a **parsimonious** (single-call) reduction that preserves the number of solutions (and provides a polynomial-time bijection on the solutions). In addition to preserving #P-hardness, parsimonious reductions preserve a stronger property called **ASP-hardness** [15], meaning that all NP search problems have a parsimonious reduction to the problem. In addition to #P-hardness, ASP-hardness implies NP-hardness of the decision problem as well as the k-ASP problem: given k solutions to an instance, find another solution. The weaker notion of ***c*-monious** reduction increases the number of solutions by exactly a factor of c. Such a reduction implies both NP-hardness and #P-hardness (but not ASP-hardness), as zero-solution instances are preserved and we can recover the answer to our initial counting problem by dividing by c.

But even a c-monious reduction from #2SAT (say) can be difficult, compared to a standard NP-hardness reduction. A c-monious reduction built from standard variable and clause gadgets must have exactly the same number of solutions to every gadget; then c is the product of these counts. In particular, a clause gadget must have exactly the same number of solutions *no matter how it is satisfied* – no matter whether it is satisfied by the first clause, the second clause, or both clauses. This property often does not come without substantial effort. For example, in Lichtenstein's reductions from (planar) 3SAT to Hamiltonicity and vertex cover [6], the number of solutions to each clause is equal to the number of true literals that satisfy the clause. In this paper, we prove that such a reduction still establishes #P-hardness.

1.1 Our Results: Generalized #SAT

Specifically, we define a framework called **Generalized #SAT**, where a clause can contribute a count of not just 0 or 1. A **clause type** is defined by a function from literal truth values to nonnegative integers, indicating the number of ways the clause is satisfied by that assignment. The number of solutions to the formula is then defined to be the product of the clause function outputs, summed over all possible assignments. In particular, if a clause function outputs zero, then that assignment still does not contribute to the number of solutions (similar to #SAT).

In Section 3, we present a complete complexity dichotomy of Generalized #SAT for the case of 2 variables per clause, precisely characterizing the computational hardness of every clause type as either in FP or #P-hard. In particular, Generalized #SAT is #P-complete for the function $f(x, y) = x + y$ counting the number of satisfying literals. Thus Lichtenstein's reductions [6], adapted to reduce from 2SAT instead of 3SAT, prove #P-hardness of Hamiltonicity and vertex cover. While these results have been proved in other ways since [9, 7],[2] it is comforting to know that existing NP-hardness reductions can be

[2] The first proof, by Valiant [12], seems to have never been published.

more easily extended to #P-hardness, potentially saving time in the future. In some cases, a c-monious clause gadget may be very difficult or even impossible to construct, while Generalized #SAT provides the flexibility necessary for a reduction from #2SAT.

1.2 Our Results: Constraint Graph Satisfiability

We show the applicability of our Generalized #SAT framework by using it to solve another open problem: analyzing the complexity of counting solutions to the Constraint Graph Satisfiability (CGS) problem [3]. A **constraint graph** is a graph, usually 3-regular, together with edge weights of 1 and 2, also referred to as edge colors red and blue respectively. Given such a constraint graph, the goal in CGS is to find an orientation of the graph (direction of the edges) such that every vertex has a total incoming weight of at least 2. Constraint graphs are the foundation of Nondeterministic Constraint Logic (the reconfiguration version of CGS according to edge reversals), which is a popular framework for proving puzzles and games PSPACE-hard [3]. CGS was shown NP-complete over 15 years ago [3, Section 5.1.3], but the complexity of its counting problem remained unsolved.

In Section 4, we prove that CGS is ASP-complete in general, implying #P-completeness of #CGS. Then we analyze subproblems of #CGS, where the graph is restricted to only certain vertex and edge types, providing an almost-complete complexity characterization for these various subproblems, as summarized in Table 1. Specifically, there are three interesting degree-3 vertices in CGS:

1. **maj** (majority) vertices have three incident red edges (at least two of which must point in to reach an incoming weight of 2);
2. **or** vertices have three incident blue edges (at least one of which must point in); and
3. **and** vertices have two incident red edges and one incident blue edge (where the blue edge can point out only if both red edges point in).

The original NP-completeness proof of CGS [3, Theorem 5.4] uses just AND and OR vertices (while other Constraint Logic proofs in [3] use MAJ vertices, under the name "CHOICE"). In addition to restricting to an arbitrary subset of these vertex types, we can consider two types of edges. In **matching edge weights**, each edge is uniformly red or blue, so both endpoints view the edge as having the same weight. In **arbitrary edge weights**, we allow an edge to have red/weight 1 at one endpoint and blue/weight 2 at the other endpoint. In [3], the latter type of edge is called a "red-blue conversion". While red-blue conversion can be simulated with matching edge weights [3, Figure 2.4] this simulation is not parsimonious, so for our analysis the problems differ.

▶ **Conjecture 1.** *There exists no c-monious* RED-BLUE *conversion gadget unless #P = FP.*

Fortunately, most reductions from Constraint Logic or CGS do not care whether edge weights are matching. Only vertex gadgets depend on their incident edge colors, while there is only one distinct edge gadget independent of color. Thus, arbitrary edge weights are of primary interest, and in this case, we provide a complete characterization of complexity. We start by showing that CGS with only AND vertices can be reduced from Generalized #SAT to show #P-hardness, while the decision problem is easy. If we have only MAJ vertices, then even counting is easy. (We leave open the case of only OR vertices.) Next, we show that when allowing all three vertex types, we can design a parsimonious reduction from Exactly 1-in-3 SAT, proving ASP- and #P-completeness. By a slight modification, the reduction can be made c-monious with only ANDs and ORs, achieving #P-completeness in this case. Finally, for all pairs of vertex types, we show that counting is easy.

Table 1 Known and new results for Constraint Graph Satisfiability (CGS). The first three columns indicate allowed gadgets (✓). The fourth column specifies whether edge weights must be matching at either endpoint, or whether they can be arbitrary on either end (equivalently, allowing red-blue conversions). ASP-c(omplete) means that the reduction is parsimonious, which implies #P-c(ompleteness) as well. All hardness results except Theorem 19 can be encoded as planar graphs.

AND	OR	MAJ	Edge Weights	CGS	#CGS
✓	✓	✓	Arbitrary	NP-c [3]	**ASP-c** (Thm. 15)
✓	✓	✓	Matching	NP-c [3]	(open)
✓	✓	×	Arbitrary	NP-c [3]	**#P-c** (Thm. 11)
✓	✓	×	Matching	NP-c [3]	(open)
×	✓	✓	Arbitrary	**P** (Thm. 17)	**#P-c** (Thm. 19)
×	✓	✓	Matching	**P** (Thm. 17)	(open)
✓	×	✓	Arbitrary	**P** (Thm. 18)	**#P-c** (Thm. 11)
✓	×	✓	Matching	**P** (Thm. 18)	**FP** (Thm. 21)
✓	×	×	Arbitrary	**P** (Thm. 18)	**#P-c** (Thm. 11)
✓	×	×	Matching	**P** (Thm. 18)	**FP** (Thm. 12)
×	✓	×	(Matching)	**P** (Thm. 17)	#P-c [1]
×	×	✓	(Matching)	**P** (Thm. 17)	**FP** (Thm. 13)

Most of our hardness results hold when restricted to *planar* constraint graphs, making them particularly amenable for reductions to games and puzzles. The only exception is the case of OR and MAJ vertices with arbitrary edge weights, for which we have been unable to build a crossover gadget.

2 Preliminaries

2.1 Generalized #SAT

We define **Generalized #SAT** as follows. Each version of Generalized #SAT is specified by a **clause type**, a nonnegative integer function $f\colon \{0,1\}^k \to \mathbb{Z}_{\geq 0}$, which describes the number of ways a given assignment of k literals satisfies the clause. We allow negations of variables, denoted with a bar (like $\bar{x}$), to be used freely in clauses. An input to Generalized #SAT consists of a number n, the number of variables (denoted $x = (x_1, \ldots, x_n)$), and a set of m clauses $C = \{\phi_1, \ldots, \phi_m\}$. Each ϕ_i is a k-tuple of literals, like $(x_{i_1}, \bar{x}_{i_2}, \ldots, x_{i_k})$. Let x_ϕ denote the restriction of a variable assignment x to the variables in a clause ϕ. The goal is to compute the number of ways to satisfy all clauses:

$$\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} \prod_{\phi\in C} f(x_\phi). \tag{1}$$

Previous work has proved that #2SAT [13] and #Max Cut [8] are #P-complete.

2.2 Constraint Graph Satisfiability

A **constraint graph node** (E, W, c) consists of a set E of incident edges, an assignment W of nonnegative weights to the edges, and a lower bound c on the total incoming weight. Usually we restrict edge weights to either 1, which we call **red**, or 2, which we call **blue**. A **constraint graph** G is a set of constraint graph nodes and edges, where each edge appears in exactly two nodes. A **configuration** of G is an assignment of directions to the edges such that, for each node (E, W, c), the total weight of incoming edges among E is at least c.

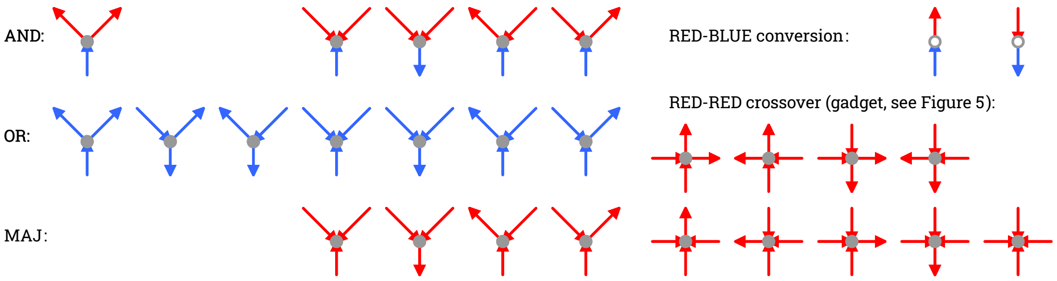


Figure 1 Overview of the Constraint Graph Satisfiability node types AND, OR, MAJ, and RED-BLUE conversion, as well as their allowed edge orientations. Thereby the total in-weight has to be at least 2 (except for RED-BLUE conversion), achievable by a single blue in-edge (weight 2) or two red in-edges (two times weight 1). The crossover gadget internally uses AND, OR, MAJ, and RED-BLUE conversion, as depicted in Figure 5.

▶ **Problem 1** (Constraint Graph Satisfiability (CGS)). *Given a constraint graph G does there exists a legal configuration?*

▶ **Problem 2** (Counting Constraint Graph Satisfiability (#CGS)). *Given a constraint graph G, how many legal configurations of G exist?*

The definitions above are more general than the standard definitions [3]. In particular, they allow the notion of edge weight to be local to a vertex, instead of being specified at the graph level. If an edge can have different weights at each end, we call the problem **arbitrary edge weights**, while if an edge is required to have the same weight at both ends, we call the problem **matching edge weights**. Equivalently, we can think of arbitrary edge weights as equivalently allowing for a **red-blue conversion** gadget – an edge that is red on one end and blue on the other.

In the standard formulation of constraint graph satisfiability, three types of degree-3 vertices: AND, OR, and MAJ. Figure 1 gives an overview of these types and all allowed edge orientations. An **and** vertex has two red edges and one blue edge, so it is satisfied only when both red edges are in-edges or the blue edge is an in-edge. Therefore, it mimics a boolean AND, where if both red edges are inward, then the vertex is "true" (the blue edge can be outward) and otherwise, it is "false" (the blue edge must be inward). An **or** vertex has three blue edges, so it is satisfied as along at least one edge is inward, similar to a boolean OR. A **maj** vertex has only red edges, so it is only satisfied if at least two (the majority) of its three edges are inward.

CGS is also a special case of Graph Orientation [4] where the valid configurations are shown in Figure 1.

In many cases, we would like the constraint graph to be planar. A useful tool for this is a CROSSING vertex allows us to build non-planar graph out of planar graphs. If two edges cross, we put a CROSSING vertex at their intersection point. This vertex mimics the standard crossover gadget used in graph reductions.

3 Generalized #SAT Dichotomy

In this section we outline our size-2 dichotomy results. We begin with some definitions that help describe the easy cases. Throughout this section, we let $f\colon \{0,1\}^2 \to \mathbb{Z}_{\geq 0}$ be a 2-variable clause type.

▶ **Definition 2.** *f is **factorable** if there exist functions g and h such that $f(x, y) = g(x)h(y)$ for all $(x, y) \in \{0, 1\}^2$.*

▶ **Definition 3.** *f is **2-color** if either $f(0, 0) = f(1, 1) = 0$ or $f(0, 1) = f(1, 0) = 0$.*

Our main theorem of this section is the following dichotomy result.

▶ **Theorem 4** (Dichotomy Theorem). *Generalized #SAT with a single 2-variable clause type f is in FP if f is either factorable or 2-color. Otherwise, it is #P-complete.*

3.1 Easy Cases

First, we describe polynomial-time algorithms for the factorable and 2-color cases.

▶ **Lemma 5.** *If f is factorable, Generalized #SAT can be solved in polynomial time.*

Proof. Let $f(x, y) = g(x)h(y)$. The product $\prod_{\phi\in C} f(x_\phi)$ can be expanded as a product of gs and hs, $\prod_{\phi\in C} f(x_\phi) = \prod_{i=1}^{n} g(x_i)^{a_i} g(\bar{x}_i)^{b_i} h(x_i)^{c_i} h(\bar{x}_i)^{d_i}$, for some exponents a_i, b_i, c_i, d_i. The sum of this expression over all $(x_1, \ldots, x_n) \in \{0, 1\}^n$ is equal to $\prod_{i=1}^{n} \sum_{x_i\in\{0,1\}} g(x_i)^{a_i} g(\bar{x}_i)^{b_i} h(x_i)^{c_i} h(\bar{x}_i)^{d_i}$, which can be evaluated in polynomial time. ◀

▶ **Lemma 6.** *If f is 2-color, Generalized #SAT can be solved in polynomial time.*

Proof. Each clause $\phi = f(x, y) \in C$ forces x and y to be either equal to each or not equal to each other. Consider a graph on n nodes one for each variable, and add an edge between x and y for all $\phi \in C$. Within each connected component of this graph, fixing an assignment to any one variable forces all the others. There are at most 2 ways to satisfy each connected component, and the answer is the product of the answers for each component independently. ◀

3.2 Hardness

Next, we prove that all remaining cases are #P-complete, we will show that all hard clause types reduce to one of the two following cases.

▶ **Definition 7** (2SAT-like). *f is **2SAT-like** if $f(0, 0) = 0$, $f(0, 1) = f(1, 0) = a > 0$, $f(1, 1) = b > 0$.*

▶ **Definition 8** (Max-Cut-like). *f is **Max-Cut-like** if $f(0, 0) = f(1, 1) = a > 0, f(0, 1) = f(1, 0) = b > 0, a \neq b$*

▶ **Lemma 9.** *If f is 2SAT-like, then Generalized #SAT is #P-complete.*

Proof. We reduce from #2SAT. For each clause $\varphi = x \vee y$ in the #2SAT instance (where x and y are literals), add a unique new variable z and three clauses $\phi_1, \phi_2, \phi_3 = f(x, y), f(\bar{x}, z), f(\bar{y}, z)$ to the generalized #SAT formula. The number of ways to satisfy the clause is 0 if $x \vee y$ is false and a^2b otherwise:

- If $(x, y) = (0, 0)$, $f(0, 0) = 0$, so $f(x, y)f(\bar{x}, z)f(\bar{y}, z) = 0$.
- If $(x, y) = (0, 1)$ or $(1, 0)$, then $\sum_{z\in\{0,1\}} f(x, y)f(\bar{x}, z)f(\bar{y}, z) = f(x, y) \sum_{z\in\{0,1\}} f(0, z)f(1, z) = a^2b$.
- If $(x, y) = (1, 1)$, then $\sum_{z\in\{0,1\}} f(x, y)f(\bar{x}, z)f(\bar{y}, z) = f(x, y) \sum_{z\in\{0,1\}} f(0, z)f(0, z) = a^2b$.

Therefore, this reduction is $(a^2b)^m$-monious, where there are m clauses in the #2SAT instance, so Generalized #SAT is #P-complete. ◀

▶ **Lemma 10** (Max-Cut-like). *If f is Max-Cut-like, Then Generalized #SAT is #P-complete.*

Proof. We reduce from #Max Cut. Let the input of a #Max Cut instance be a graph $G = (V, E)$, and calculate the number $M := 1 + \lceil \log_{\max(a/b, b/a)}(2^{|V|}) \rceil \in O(|V|)$. Associate a boolean variable to each vertex in V. For each edge $(x, y) \in E$, add M clauses of the form $f(x, y) f(\bar{x}, \bar{y})$. The answer to this Generalized #SAT instance is

$$N := \sum_{S \sqcup T = V} b^{M \text{cut}(S,T)} a^{M(|E| - \text{cut}(S,T))} = \sum_{c=0}^{|E|} k_c b^{Mc} a^{M(|E|-c)},$$

where $\text{cut}(S, T) := \#\{(u, v) \in E \mid u \in S, v \in T\}$ and $k_c := \#\{(S, T) \mid S \sqcup T = V, \text{cut}(S, T) = c\}$. Note that $0 \le k_c \le 2^{|V|}$ and the ratios of adjacent coefficients $\frac{b^{Mc} a^{M(|E|-c)}}{b^{M(c+1)} a^{M(|E|-(c+1))}} = \left(\frac{a}{b}\right)^M$ differ by more than $2^{|V|}$. Therefore, it is possible to exactly extract all the numbers $\{k_c\}_{0 \le c \le |E|}$ from N, and the answer to #Max Cut is $k_{\max(c\colon k_c > 0)}$. ◀

3.3 Main Dichotomy Result

We now present the complete proof of our size-2 dichotomy, based on our four clause types each defined in relevant theorems, factorable, 2-colorable for easy cases, and #2SAT-like and Max-Cut-like for the hard cases.

Proof of Theorem 4. If f is factorable or 2-colorable, Generalized #SAT is in FP by Lemma 5 or Lemma 6. Suppose f is not one of these cases. Then at most one of the values $f(x, y)$ for $(x, y) \in \{0, 1\}^2$ can be 0.

If one of these values is 0, we reduce from Lemma 9. By negating one or both arguments of f, without loss of generality we let $f(0, 0) = 0$ and $f(0, 1), f(1, 0), f(1, 1) > 0$. Replace each 2SAT-like clause $f_2(x, y)$ with two clauses, $f(x, y) f(y, x)$.

If none of these values is 0, we reduce from Lemma 10. Replace each Max-Cut-like clause with two clauses, $f(x, y) f(\bar{x}, \bar{y})$. ◀

4 Counting Constraint Graph Configurations (#CGS)

Having established the generalized #SAT dichotomy, we now use these results as a tool to prove hardness of #CGS. Specifically, we establish the many results of Table 1, which we hope will enable further use of CGS for many puzzles and games. First, we establish easy cases with single vertex types and also prove that this problem is already #P-complete on graphs using AND nodes with general edge weights. Then, in Section 4.2 we prove ASP-hardness when allowed all three vertex types – AND, OR, and MAJ– and general edge weights. Finally we restrict the set of allowed vertex types, and present bounds and complexity characterizations when counting solutions is hard.

4.1 Constraint Graphs with a Single Vertex Type

We show that #CGS on graphs with just AND vertices in #P-complete with general edge weights. But if we enforce matching edge weights, then #CGS is in FP. We also show that for just MAJ vertices, #CGS is in FP. (Because MAJ vertices have just one edge type, the notion of matching or general edge weights is irrelevant.)

To show that AND vertices with general edge weights is #P-complete, we reduce from #SAT. In the following, we design a (parsimonious) variable gadget for every variable x, as depicted in Figure 2. The gadget uses $O(k)$ nodes where k is the number of times x appears,

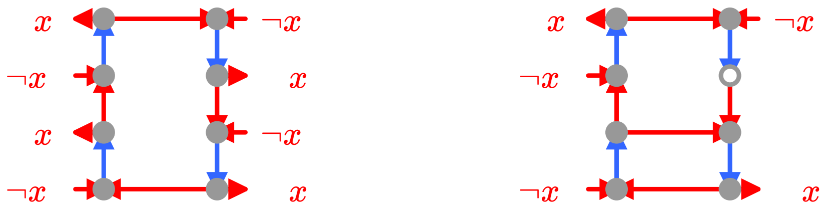


Figure 2 Variable gadget (left) showing an equal number of positive and negative variable occurrences. Using RED-BLUE conversion vertex types, we can construct an odd number of out-going edges (right). Note that at most one red-blue conversion is needed (in case of different parity of x and $\neg x$ occurrences), as x's and $\neg x$'s can be linked by a red edge (depicted above).

thus each vertex functions as a literal, with the edge orientation propagating the value. Every variable occurrence is thereby connected to an AND node such that if the edge is directed towards x ($\neg x$) the literal x ($\neg x$) evaluates to true. By construction, the variable gadget has only two legal edge orientations. One of which is shown in Figure 2 (x is true), the other configuration is its complement, where all edges are inverted (x is false).

This gadget and Section 3 are sufficient ingredients to establish the following tight result.

▶ **Theorem 11** (#P-Hardness). *Counting with just AND is #P-complete if we do not enforce matching edge weights.*

Proof. We reduce from 2SAT-like Generalized #SAT with clause type $f(x, y) = x + y$ (see Lemma 9) and construct a #CGS instance as follows. We reuse the variable gadget of Figure 2. However, the clause gadget for a binary clause $c = (\ell \vee \ell')$ (ℓ, ℓ' are two distinct literals) is simply a red edge connecting literal ℓ to literal ℓ'. Indeed, by construction, the clause gadget requires that the edge is directed either to ℓ (i.e. ℓ is true) or to ℓ' (i.e. ℓ' is true). Consequently, if both ℓ and ℓ' are true, clause c is considered with weight 2, as we can always direct the edge from ℓ to ℓ' or from ℓ' to ℓ. This is as required by Equation (1), which coincides with the number of CGS solutions of the constructed instance and establishes the result. ◀

This result immediately works for planar graphs as well, as in Lemma 9 we can alternatively reduce from planar (monotone) #2SAT, which is #P-complete [11]. This result is tight, as the counting problem #CGS is easy for just AND nodes if we enforce matching edge weights.

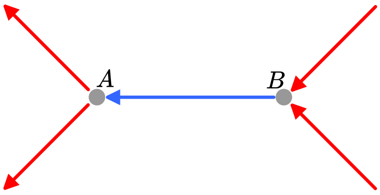


Figure 3 AND vertices must pair along blue edges if matching edge weights are enforced.

▶ **Theorem 12.** *#CGS with just AND with matching edge weights is in FP.*

Proof. Note that since matching edge weights is enforced, each AND vertex must pair off with exactly one other AND vertex. They connect to each other via their blue input and form a "super-vertex" with four red inputs/outputs (Figure 3). For any such pair of AND vertices, A and B, clearly, their shared blue edge can only be directed into one of them. Without loss of generality, assume it is directed into A. Then, for B to be satisfied, the red edges of B must point into itself. Note that this implies each degree 4 super-vertex must have in-degree at least 2. Our assumption of the blue edge pointing into A fixes the direction of the red edges of B. It also forces that the red edges of A must point outwards. Assume one of A's red edges points inwards; this super vertex has in-degree 3. This implies there must be some other corresponding super vertex with an in-degree of exactly 1, which is unsatisfied. Therefore, for any pair of AND vertices, if we set the direction of their shared blue edge, the direction of their red edges is forced, and those edges then force the direction of the edges of other pairs of ANDs, and so forth. Therefore, a connected graph of AND vertices with matching edge weights has either no solutions or two solutions (we can reverse the edges of one solution to get a second). Hence, for a graph of AND vertices with matching edge weights with k connected components, the number of solutions is either 0 or 2^k. ◀

Additionally, #CGS on a graph of just MAJ vertices is also in P, by a relatively simple proof that such a graph is never satisfied.

▶ **Theorem 13.** *#CGS with just MAJ is in FP.*

Proof. MAJ vertices require an in-degree of 2 to be satisfied. However, a graph of only MAJ vertices is 3-regular. Therefore, the average in-degree is exactly 1.5 so at least one vertex must have in-degree less than two. Therefore, there are always 0 solutions. ◀

For the case of only ORs, an equivalent version of #3SAT was shown to be #P-complete [1]. We can view each OR vertex as a 3CNF clause and each edge as a variable that appears once with each sign, i.e., for each variable x, there exists exactly one x literal and exactly one $\overline{x}$ literal in the formula.[3]

▶ **Theorem 14** ([1])**.** *#CGS with just OR is #P-complete even when restricted to planar graphs.*

4.2 Parsimonious Reduction from #1-in-3SAT to #CGS

First, we discuss a parsimonious reduction that uses AND, OR, MAJ, and RED-BLUE conversion vertex types. This reduction then yields ASP-hardness for CGS, which makes this formalism a perfect tool to prove that for puzzles and games, finding a second solution is still hard. To this end, we directly reuse the variable gadget of Figure 2. Figure 4 (left) depicts the clause gadget for the clause $x \vee \neg y \vee z$, which works via the three 1-in-3SAT cases. Both gadgets are then used in the reduction below.

▶ **Theorem 15** (ASP-hardness)**.** *The another-solution problem for CGS is ASP-hard, even if restricted to the vertex types AND, OR, and MAJ.*

Proof. The reduction consisting of variables gadgets (Figure 2) and clause gadgets (Figure 4) is correct. By construction, the variable gadget for a variable x in Figure 2, which is attached at the bottom, prevents that there are both outgoing x and $\neg x$ edges (simultaneously). The clause gadget for a clause c shown in Figure 4 ensures that precisely one of the cases c_1, c_2, c_3 holds. Indeed, each pairwise combination of the three cases is by construction contradicting. However, in a solution every edge direction is pinned down as either the case c_i holds (outgoing blue edge, which requires all outgoing red edges towards c_i), or

[3] This version of #3SAT is named #Pl-Rtw-Opp-3CNF.

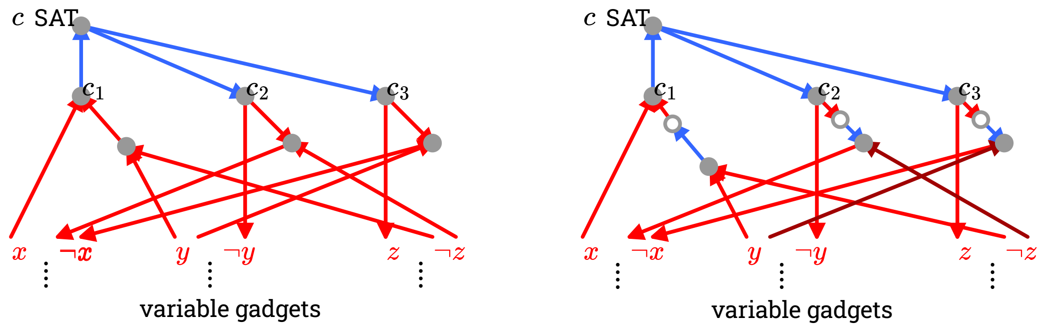


Figure 4 Clause gadget (left), where for a 1-in-3SAT clause of the form $c = x \vee \neg y \vee z$ we parsimoniously preserve solutions by expressing three cases (terms) $c_1 = (x \wedge (y \wedge \neg z))$, $c_2 = (\neg y \wedge (\neg x \wedge \neg z))$, $c_3 = (z \wedge (\neg x \wedge y))$, where the first literal of each c_i is the one from c being true (and the two remaining literals occur negated in c). Indeed these three cases allow us to preserve a bijective relationship between 1-in-3SAT solutions and satisfying edge orientations. However, it is crucial that the brackets are precisely as above, as this pins down edge orientations for MAJ vertex types connected to c_i. Roughly, in c_i the literals of c that are supposed to be false are connected by a MAJ vertex. If we replace the MAJ vertices by AND vertices (right), the orientation of the dark red edges is free. While this does not preserve parsimony, for m clauses the reduction is still 4^m-monious.

there is precisely one outgoing edge of the MAJ vertex. Indeed, by construction the pairwise intersections of literals in c_i, c_j for $i \neq j$ are of size 1. Hence, if c_j does not hold then exactly one of the two literals of c_j that are negated in c are true. This literal therefore is a predecessor of the MAJ vertex attached to c_j.

Since this reduction is parsimonious and 1-in-3SAT is ASP-hard [15], we conclude the result. ◀

Observe from the variable gadget in Figure 2 that if for every variable x, the number of occurrences of x and $\neg x$ are identical, RED-BLUE conversion vertex types are not needed. However, if we do not use MAJ vertex types, we need RED-BLUE conversion in the clause gadget, as depicted in Figure 4 (right). Further, this leaves 2 free edges per clause, resulting in a reduction that is $2^{2m} = 4^m$-monious.

Crossover Gadget for Planarity. Theorem 15 immediately works for planar graphs, since we can construct a parsimonious crossover gadget as depicted in Figure 5 and eliminate all crossings with a gadget.

▶ **Corollary 16** (ASP-hardness for Planar CGS). *The another-solution problem for CGS is ASP-hard, even if restricted to planar graphs over the vertex types* AND, OR, *and* MAJ*.*

Proof. Figure 5 depicts a planar red-red crossing gadget. Indeed we only need to eliminate red-red crossings as there is no face completely built out of blue edges (see Figures 2 and 4). Observe further that the gadget in Figure 5 is parsimonious, i.e., there is no free edge, assuming the original north/south and east/west edges are fixed as well. Therefore the result follows from Theorem 15. ◀

4.3 Constraint Graphs with Two Vertex Types

We now discuss what happens when our graph consists of exactly two vertex types. First, we establish easiness for the decision problems. Then, we consider counting.

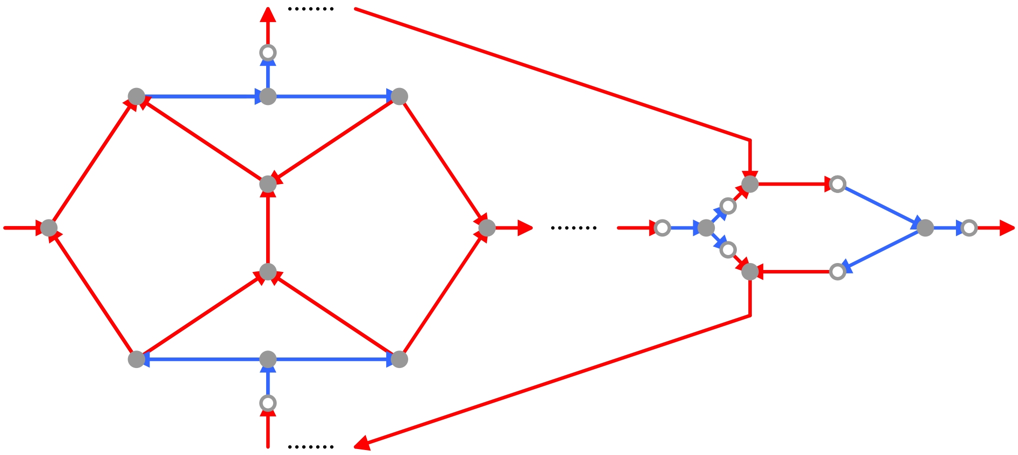


Figure 5 Parsimonious red-red crossover gadget that consists of a leaky main part (left) that is leaky in the sense that it still allows the case where both north/south vertices are directed inward and east/west are directed outward. In order to fix this, one can add a degree-4-vertex type simulation gadget (right), as shown. This gadget simulates a degree-4-vertex requiring a total in-flow of weight at least 2 if there shall be out-flow (as depicted).

4.3.1 Decision Easiness

For the decision problem CGS we obtain the following easiness results.

▶ **Theorem 17.** *CGS with* OR *and* MAJ *is in P.*

Proof. We may reduce from CGS over OR and MAJ vertices to the Max Flow problem, which is known to be in P. Each OR vertex will be replaced by sink of weight 1 and each MAJ will be replaced by a sink of weight 2. At the center of each edge add a source of weight 1.

A flow which satisfies all the sinks can be used to assign orientations on the edges of the constraint graph. The direction taken by edges out of the source is in the direction of the edge in the CGS solution. Since each sink has a weight equal to the number of incoming edges needed by the constraint node each node will be satisfied. A set of edge orientation which satisfy each edge can be used to assign the flow of each edge in the same way. ◀

▶ **Theorem 18.** *CGS for* AND *and* MAJ *is in P.*

Proof. We reduce to 2-SAT, which is in P [5]. Note that an AND vertex with inputs labeled b, r_1, r_2, where b is its blue input, and r_1, r_2 are its red inputs, can be represented by the boolean equation $(b) \vee (r_1 \wedge r_2) = (r_1 \vee b) \wedge (r_2 \vee b)$, where a variable is true if its corresponding edge is directed into the vertex. Additionally, a MAJ vertex with inputs a, b, c can be represented by $(a \wedge b) \vee (a \wedge c) \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \wedge (b \vee c)$. So each vertex can be represented with a set of 2-SAT clauses. Note that each variable represents an edge that connects two vertices. Therefore, each variable will only appear in sets of clauses corresponding to the two vertices it is incident to. We negate a set of these literals so that only one may be true to simulate the edge only pointing in one direction. Therefore we can write any constraint graph of AND and MAJ vertices as a 2-SAT formula. ◀

4.3.2 #CGS Hardness

In this section we show that, if we have exactly two vertex types (AND/OR, AND/MAJ, or OR/MAJ) and do not enforce matching edge weights, then #CGS is hard.

▶ **Theorem 19** (#P-Hardness for Two Vertex Types)**.** *#CGS is #P-hard if we do not enforce matching edge weights and are restricted to* AND/OR *vertices,* AND/MAJ *vertices, or* OR/MAJ *vertices, with at least one vertex of each of the two types.*

Note that #P-hardness for the cases including AND vertices follows already from Theorem 11. Consequently, it suffices to establish the following lemma.

▶ **Lemma 20.** *#CGS with* OR *and* MAJ *is #P-complete, even when there exists a connected component which contains both vertex types.*

Proof. We reduce from counting the number of perfect matchings in a 3-regular bipartite graph $(V = \{O \bigcup M\}, E)$, which has been shown to be #P-hard [2]. We will replace one partition of vertices $\boldsymbol{O}$ with OR vertices and the other $\boldsymbol{M}$ with MAJ.

The set of edges in the matching correspond to edges directed from a node in $\boldsymbol{M}$ to a node in $\boldsymbol{O}$. Each $\boldsymbol{O}$ node requires edge pointed in to satisfy its inflow. Each $\boldsymbol{M}$ node can only be used in a single matching as it requires 2 edges pointed in. ◀

4.3.3 #CGS Easiness

However, with matching edge weights #CGS is in FP if we have two vertex types, one of which must be MAJ. This further strengthens Theorems 17 and 18.

▶ **Theorem 21** (Counting is easy MAJ)**.** *#CGS can be solved in polynomial time if we enforce matching edge weights and are restricted to two vertex types, one of which is* MAJ *(either* MAJ/OR *or* MAJ/AND*).*

Each pair of vertices has a separate proof, hence we prove this theorem via the following lemmas.

▶ **Lemma 22.** *#CGS can be done in polynomial time if we enforce matching edge weights and are restricted to only* MAJ *and* OR *vertices, and there is a nonzero number of* MAJ *vertices.*

Proof. Since we enforce matching edge weights, OR and MAJ vertices cannot connect to each other as OR vertices only take blue inputs, and MAJ vertices only take red inputs. Therefore, any constraint graph with OR and MAJ will have multiple components, ones made up of only OR vertices and ones made up of only MAJ vertices. By Theorem 13, we know each MAJ component can never be satisfied; hence, the number of solutions to CGS with only MAJ and OR vertices with matching edge weights is always 0. ◀

Note that in Lemma 22 we require the number of MAJ vertices to be nonzero as otherwise the graph has only OR vertices, which we leave an open problem, as discussed in section 4.1.

▶ **Lemma 23.** *#CGS can be done in polynomial time if we enforce matching edge weights and are restricted to only* MAJ *and* AND *vertices.*

Proof. By enforcing matching edge weights, an argument similar to Theorem 12 applies, as MAJ vertices have no blue inputs; therefore, each AND vertex must pair with another AND through a blue edge. Note that the average in-degree in each of these AND pairs is ≥ 1.5. Further, each MAJ vertex requires in-degree > 2 to be satisfied. Therefore, if a constraint graph contains AND and MAJ vertices, the average in-degree must be > 1.5, or else it is not satisfied. But this is impossible; if the graph is 3-regular, the average in-degree is 1.5. Hence, if there is a nonzero number of MAJ vertices, the number of solutions is 0. Otherwise Theorem 12 applies and else the number of solutions is either again 0 or 2^k where k is the number of components of the graph. ◀

5 Conclusion and Future Work

In this work we presented a novel generalized #SAT framework as well as a dichotomy for two variables. These results then serve as the basis for novel insights into the counting complexity of graph orientation problems (constraint graph satisfiability), where we discuss an almost-complete classification (see also Table 1). We expect that counting solutions to constraint graph satisfiability (#CGS) is an interesting source to support the development and characterization of challenging puzzles, riddles, and games. Indeed, given our insights we expect many further insights into counting solutions and solving another-solution problems.

Based on our dichotomy result for 2-variable clauses, we conjecture that Generalized #SAT is in FP if f factors into a product of single-variable and 2-color functions, or it is a multiple of an affine function, and is #P-complete in all other cases. Our understanding is that for any given f, it is probably not difficult to prove this via taking “slices” with fewer variables, but we lack a systematic method to prove the dichotomy for all f that does not rely on ad-hoc casework.

References

1 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holographic reduction, interpolation and hardness. *Computational Complexity*, 21:573–604, 2012. doi:10.1007/S00037-012-0044-6.

2 Paul Dagum and Michael Luby. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102(2):283–305, 1992. doi:10.1016/0304-3975(92)90234-7.

3 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. CRC Press, 2009.

4 Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki, and Ryuhei Uehara. Complexity of tiling a polygon with trominoes or bars. *Discrete & Computational Geometry*, 58(3):686–704, October 2017. doi:10.1007/s00454-017-9884-9.

5 M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967. doi:10.1002/malq.19670130104.

6 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.

7 MIT Hardness Group, Josh Brunner, Lily Chung, Erik D. Demaine, Della Hendrickson, and Andy Tockman. ASP-completeness of Hamiltonicity in grid graphs, with applications to loop puzzles. In *Proceedings of the 12th International Conference on Fun with Algorithms*, pages 30:1–30:20, 2024. doi:10.4230/LIPICS.FUN.2024.23.

8 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983. doi:10.1137/0212053.

9 Takahiro Seta. The complexities of puzzles, Cross Sum, and their Another Solution Problems (ASP). Senior thesis, University of Tokyo, 2002. URL: https://web.archive.org/web/20221007013910/www-imai.is.s.u-tokyo.ac.jp/~seta/paper/senior_thesis/seniorthesis.pdf.

10 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

11 Salil P. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM J. Comput.*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.

12 Leslie G. Valiant. A polynomial reduction from satisfiability to Hamiltonian circuits that preserves the number of solutions. Manuscript, 1974.

13 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

14 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

15 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 86(5):1052–1060, 2003. URL: http://search.ieice.org/bin/summary.php?id=e86-a_5_1052.

Single Family Algebra Operation on BDDs and ZDDs Leads to Exponential Blow-Up

Kengo Nakamura
NTT Communication Science Laboratories, Kyoto, Japan

Masaaki Nishino
NTT Communication Science Laboratories, Kyoto, Japan

Shuhei Denzumi
NTT Communication Science Laboratories, Kyoto, Japan

Abstract

Binary decision diagram (BDD) and zero-suppressed binary decision diagram (ZDD) are data structures to represent a family of (sub)sets compactly, and it can be used as succinct indexes for a family of sets. To build BDD/ZDD representing a desired family of sets, there are many transformation operations that take BDDs/ZDDs as inputs and output BDD/ZDD representing the resultant family after performing operations such as set union and intersection. However, except for some basic operations, the worst-time complexity of taking such transformation on BDDs/ZDDs has not been extensively studied, and some contradictory statements about it have arisen in the literature. In this paper, we show that many transformation operations on BDDs/ZDDs, including all operations for families of sets that appear in Knuth's book, cannot be performed in worst-case polynomial time in the size of input BDDs/ZDDs. This refutes some of the folklore circulated in past literature and resolves an open problem raised by Knuth. Our results are stronger in that such blow-up of computational time occurs even when the ordering, which has a significant impact on the efficiency of treating BDDs/ZDDs, is chosen arbitrarily.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Binary decision diagrams, family of sets, family algebra

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.52

Related Version *Full Version*: https://arxiv.org/abs/2403.05074

Funding This work was generally supported by JSPS KAKENHI Grant Number JP20H05963 and JST CREST Grant Number JPMJCR22D3.
Shuhei Denzumi: was supported by JSPS KAKENHI Grant Number JP23H04391.

Acknowledgements We thank Hiromi Emoto and Shou Ooba for pointing out the issues regarding the complexity of performing family algebra operations on ZDDs. We also thank Shin-ichi Minato, Jun Kawahara, and Norihito Yasuda for valuable discussions on this topic. I am grateful to the reviewers of ISAAC for the comments to improve the manuscript.

1 Introduction

Combinatorial problems, i.e., the problems dealing with combinations of a set, frequently arise in several situations such as operations research, network analysis, and LSI design. In solving such problems, it is often convenient to consider the set of combinations, i.e., the *family of (sub)sets*. For example, many combinatorial optimization problems can be formulated as selecting the best combination (subset) from the family of sets satisfying constraints. However, the number of sets in a family is possibly exponential, precluding us from explicitly retaining the family of sets.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 52; pp. 52:1–52:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To alleviate this issue, we can use *binary decision diagram* (*BDD*) [2] or *zero-suppressed binary decision diagram* (*ZDD*) [14] that is a variant of BDD. BDD and ZDD are data structures that compactly represent a Boolean function and a family of sets, respectively. Since a Boolean function f can be regarded as a family of sets by considering the set of assignments of input Boolean variables that evaluates f to *true*, BDD can also be regarded as a succinct representation of a family of sets. Moreover, they support many queries about the represented family of sets, e.g., counting the number of sets and performing linear optimization over the family. Thus, BDD and ZDD can be used as succinct indexes for a family of sets.

BDDs and ZDDs also support a number of transformation operations. For example, when we have two BDDs representing two families of sets, we can construct a BDD representing the set union of them without extracting each set from the input families. Using such operations, we can construct a BDD or a ZDD representing the desired family of sets. By collecting such transformation operations, Minato [15] considered an algebraic system called *unate cube set algebra*, whose element is a family of sets. After that, many operations were introduced, and now the system is widely called *family algebra*, whose name was given by Knuth [13]. With the algorithms performing operations on BDDs and ZDDs, every operation in the family algebra provides a useful way to construct a BDD or a ZDD representing the desired family of sets in many applications. Many of these operations have been implemented in standard BDD and ZDD manipulation packages [8, 18], and they are used in a wide range of applications, including formal verification of circuits [7, 10], analyses of power distribution networks [9, 19], and data mining [16].

However, the complexity of performing family algebra operations on BDDs and ZDDs has not been well studied, except for basic set operations. This is because some operations require complicated recursion procedures that make complexity analysis difficult. In particular, revealing *worst-case* time complexity is important to us. If the worst-case time complexity is large, it takes an unexpectedly long time to carry out even a *single* operation for certain kinds of input. If so, we should pay attention to the possibility of such input when we use BDDs and ZDDs as a way to implement the manipulation of families of sets. Therefore, we investigated the worst-case time complexity of executing a single family algebra operation on BDDs and ZDDs. Since it is known that, as described later, the sizes of a BDD and a ZDD representing the same family of sets differ in only a linear factor, this paper mainly focused on the complexity of ZDDs. After that, we mention the complexity on BDDs.

1.1 Related Work

Since the invention of ZDD [14], many family algebra operations have been proposed. Table 1 lists basic operations. As related work, we first describe the origins of these operations.

The first four operations in Table 1 are the most fundamental set operations set described by Minato [14]. The join, quotient, and remainder operations appeared in Minato's next paper [15], where the join operation is called "product" because a join can be considered to be the multiplication of two families when we view the union operation as an addition operation. These operations are peculiar to the families of sets and also fundamental in defining other family algebra operations. Later, the disjoint join and joint join operations were proposed by Kawahara et al. [12] through an extension of the join; their usage is to implicitly enumerate all of the subgraphs having a particular shape.

Restrict and permit operations were originally proposed by Coudert et al. [5], where they were called SupSet and SubSet and used for solving set cover problems or performing logic circuit minimization. The names "restrict" and "permit" come from a study by Okuno et

Table 1 List of operations on family algebra.

Operation	Definition	Is polytime in DD sizes?
Union $\mathcal{F} \cup \mathcal{G}$	$\{S \mid S \in \mathcal{F} \vee S \in \mathcal{G}\}$	Yes [14]
Intersection $\mathcal{F} \cap \mathcal{G}$	$\{S \mid S \in \mathcal{F} \wedge S \in \mathcal{G}\}$	Yes [14]
Difference $\mathcal{F} \setminus \mathcal{G}$	$\{S \mid S \in \mathcal{F} \wedge S \notin \mathcal{G}\}$	Yes [14]
Symmetric difference $\mathcal{F} \oplus \mathcal{G}$	$(\mathcal{F} \setminus \mathcal{G}) \cup (\mathcal{G} \setminus \mathcal{F})$	Yes [14]
Join $\mathcal{F} \sqcup \mathcal{G}$	$\{F \cup G \mid F \in \mathcal{F}, G \in \mathcal{G}\}$	**No** (Theorem 7)*
Disjoint join $\mathcal{F} \dot{\bowtie} \mathcal{G}$	$\{F \cup G \mid F \in \mathcal{F}, G \in \mathcal{G}, F \cap G = \emptyset\}$	**No** (Theorem 7)
Joint join $\mathcal{F} \hat{\bowtie} \mathcal{G}$	$\{F \cup G \mid F \in \mathcal{F}, G \in \mathcal{G}, F \cap G \neq \emptyset\}$	**No** (Theorem 7)
Meet $\mathcal{F} \sqcap \mathcal{G}$	$\{F \cap G \mid F \in \mathcal{F}, G \in \mathcal{G}\}$	**No** (Theorem 7)*
Delta $\mathcal{F} \boxplus \mathcal{G}$	$\{F \oplus G \mid F \in \mathcal{F}, G \in \mathcal{G}\}$	**No** (Theorem 7)*
Quotient $\mathcal{F} \,/\, \mathcal{G}$	$\{S \mid \forall G \in \mathcal{G} : S \cup G \in \mathcal{F} \wedge S \cap G = \emptyset\}$	**No** (Theorem 9)
Remainder $\mathcal{F} \,\%\, \mathcal{G}$	$\mathcal{F} \setminus (\mathcal{G} \sqcup (\mathcal{F} \,/\, \mathcal{G}))$	**No** (Theorem 9)
Restrict $\mathcal{F} \triangle \mathcal{G}$	$\{F \in \mathcal{F} \mid \exists G \in \mathcal{G} : G \subseteq F\}$	**No** (Theorem 10)*
Permit $\mathcal{F} \oslash \mathcal{G}$	$\{F \in \mathcal{F} \mid \exists G \in \mathcal{G} : F \subseteq G\}$	**No** (Theorem 10)
Nonsuperset $\mathcal{F} \searrow \mathcal{G}$	$\{F \in \mathcal{F} \mid \forall G \in \mathcal{G} : G \nsubseteq F\}$	**No** (Theorem 10)
Nonsubset $\mathcal{F} \nearrow \mathcal{G}$	$\{F \in \mathcal{F} \mid \forall G \in \mathcal{G} : F \nsubseteq G\}$	**No** (Theorem 10)
Maximal $\mathcal{F}^{\uparrow}$	$\{F \in \mathcal{F} \mid \forall F' \in \mathcal{F} : F \subseteq F' \Rightarrow F = F'\}$	**No** (Theorem 11)
Minimal $\mathcal{F}^{\downarrow}$	$\{F \in \mathcal{F} \mid \forall F' \in \mathcal{F} : F' \subseteq F \Rightarrow F = F'\}$	**No** (Theorem 11)
Minimal hitting set $\mathcal{F}^{\sharp}$	$\{S \mid \forall F \in \mathcal{F} : S \cap F \neq \emptyset\}^{\downarrow}$	**No** (Theorem 12)
Closure $\mathcal{F}^{\cap}$	$\{\bigcap_{S \in \mathcal{F}'} S \mid \mathcal{F}' \subseteq \mathcal{F}\}$	**No** (Theorem 12)

*Previous studies [17, 13] stated that they can be performed in worst-case polynomial time.

al. [17]. Later, nonsuperset, nonsubset, maximal, and minimal operations were introduced by Coudert [4] to solve various optimization problems on graphs. Furthermore, meet, delta, minimal hitting set, and closure operations were introduced by Knuth [13, §7.1.4 Ex.203,236,243] to solve various graph problems. Table 1 contains all of the transformation operations for families of sets that appeared in Knuth's book [13, §7.1.4 Ex. 203,204,236,243].

Compared to the operations themselves, the time complexity of performing them on ZDDs has not been well investigated. Minato [14] proved that the first four operations in Table 1 can be performed in polynomial time with respect to the size of input ZDDs. However, the complexity of a join operation, the most basic one among the rest, has not been fully clarified. Knuth [13, §7.1.4 Ex. 206] claimed that join, as well as meet and delta, can be performed in worst-case polynomial time, but this claim lacks proof. Conversely, Kawahara et al. [12] suggested that join, as well as disjoint join and joint join, take worst-case exponential time, again without proof. In addition to those reports, Okuno et al. [17] claimed that restrict can be performed in polynomial time, but they used the unproven proposition that join can be performed in polynomial time. Furthermore, Knuth [13, §7.1.4 Ex. 206] stated that the worst-case complexity of the quotient operation was an open problem.

1.2 Our Contribution

In this paper, we prove that, for the operations in Table 1 aside from the first four operations, there exist polynomial-sized ZDDs such that after taking the operation, the ZDD size becomes exponential. For example, for the join operation, we prove that there exist sequences of families of sets $\{\mathcal{F}_m\}$ and $\{\mathcal{G}_m\}$ such that the ZDD sizes representing $\mathcal{F}_m$ and $\mathcal{G}_m$ are polynomial in m, while the ZDD size representing $\mathcal{F}_m \sqcup \mathcal{G}_m$ is exponential in m. This result implies that these operations cannot be performed in worst-case polynomial time with respect to the size of input ZDDs. Thus, we refute the statement raised by Knuth [13] and Okuno et al. [17] that join, meet, delta, and restrict can be performed in worst-case polynomial time.

We also resolve the worst-case complexity of the quotient operation. Moreover, we also prove that the operations in Table 1, except for the first four operations, cannot be performed in polynomial time even when families are represented by BDDs. Since Table 1 contains all the family algebra operations raised by Knuth [13], this paper concludes what kind of family algebra operations can be performed in polynomial time on BDDs and ZDDs.

Our result is stronger in that the resultant BDD/ZDD's size remains exponential for any *order of elements*. BDD/ZDD structures follow a total order of the elements in the base set, and it is known that this element order has a significant impact on the BDD/ZDD size. For example, it is known that a multiplexer function can be represented in linear-sized BDD by managing the ordering while its size becomes exponential when the ordering is terrible [13, p.235]. However, we also prove that for the sequences used in proving the above, the resultant BDD/ZDD's size is exponential in m regardless of the order of elements. This suggests that we cannot shrink the BDD/ZDD size after taking an operation by managing the element order. Some famous BDD manipulation packages such as CUDD [18] implemented dynamic reordering, the reordering of elements after executing operations to shrink the BDD/ZDD size and thus increase the efficiency of BDD/ZDD manipulations. Nevertheless, our results suggest that the worst-case complexity of carrying out operations cannot be polynomial, even if we employ dynamic reordering.

Note that this follows the research line of Bollig [1] as follows. Yoshinaka et al. [20] refuted Bryant's conjecture, which is about the complexity of performing operations on BDDs, but their counterexample was somewhat weak in that the order of elements they used was unfavorable for BDD representations. Bollig [1] later resolved this issue by proposing simpler counterexamples. Similar to this, our results imply that the exponential blow-up in taking an operation on BDDs/ZDDs occurs not only when the order of elements is unfavorable but also when it is good for BDD/ZDD representations.

From the viewpoint of applications, BDDs/ZDDs are usually built by applying multiple family algebra operations in combination with some direct construction methods such as Simpath [13] and frontier-based search [11], which are fixed-parameter tractable algorithms with pathwidth. However, the number of required operations stays constant in many applications. If every operation can be performed in polynomial time, we can enjoy the polynomial time complexity in BDD/ZDD sizes even for these applications. However, our results suggest this is not the case except for the first four operations. In addition, although we rely on specific input examples to prove non-polynomial lower bounds, we later discuss that such blow-up may occur for other input; the detailed discussions are in Section 3.5. Therefore, our theoretical results have practical importance.

2 Preliminaries

2.1 Zero-suppressed Binary Decision Diagram

A *zero-suppressed binary decision diagram* (*ZDD*) [14] is a rooted directed acyclic graph (DAG)-shaped data structure for representing a family of sets. First, we describe the structure of ZDD. ZDD $\mathtt{Z}$ consists of node set $\mathtt{N}$ and arc set $\mathtt{A}$, where the node set contains *terminal* nodes $\top, \bot$ and other internal nodes. Terminal nodes have no outgoing arcs, while every internal node has two outgoing arcs called *lo-arc* and *hi-arc*. The nodes pointed by the lo-arc and the hi-arc outgoing from a node $\mathtt{n}$ are called *lo-child* $\mathsf{lo}(\mathtt{n})$ and *hi-child* $\mathsf{hi}(\mathtt{n})$ of $\mathtt{n}$. Every internal node $\mathtt{n}$ is associated with an element called *label* that is denoted by $\mathsf{lb}(\mathtt{n})$. ZDDs must follow the *ordered property*: Given a total order of elements $<$, the label of the parent

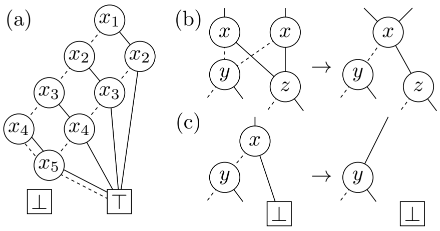


Figure 1 (a) Example of a ZDD representing the family of subsets of $\{x_1, \dots, x_5\}$ such that the cardinality is less than 3. (b) Schematic of node sharing. (c) Schematic of zero suppression.

node must precede that of the child node, i.e., $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{lo}(\mathtt{n}))$ and $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{hi}(\mathtt{n}))$ must hold for every internal node $\mathtt{n}$. Note that the child node is always allowed to be a terminal node. Finally, the size of a ZDD is defined by its number of nodes.

Next, we describe the semantics of ZDD.

▶ **Definition 1.** *For ZDD node* $\mathtt{n}$*, the family* $\mathcal{F}_{\mathtt{n}}$ *of sets represented by* $\mathtt{n}$ *is defined as follows. (i) If* $\mathtt{n} = \top$*, then* $\mathcal{F}_{\mathtt{n}} = \{\emptyset\}$*. (ii) If* $\mathtt{n} = \bot$*, then* $\mathcal{F}_{\mathtt{n}} = \emptyset$*. (iii) Otherwise,* $\mathcal{F}_{\mathtt{n}} = \mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup (\{\{\mathsf{lb}(\mathtt{n})\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{n})})$*. Furthermore, the family of sets represented by* $\mathtt{Z}$ *is that represented by root node* $\mathtt{r}$*, where the root node is the only node having no incoming arcs.*

Note that $\{\emptyset\}$ and $\emptyset$ are different families; the former is the family consisting of only an empty set, while the latter is the family containing no set. For example, Figure 1a is the ZDD representing the family of subsets of $\{x_1, \dots, x_5\}$ whose cardinality is less than 3. Solid and dashed lines represent hi- and lo-arcs, and the element inside a circle indicates its label.

Without restrictions on the structure, there exist many ZDDs representing the same family of sets. However, by imposing restrictions, we can obtain a *canonical* ZDD, i.e., an identical ZDD structure, for every family of subsets. This canonical form is called *reduced ZDD*, and a reduced ZDD can be obtained from any ZDD by repetitively applying the following two rules. The first rule is *node sharing*: If there exist two nodes $\mathtt{n}$ and $\mathtt{m}$ whose lo-child, hi-child, and label are equal, we merge these two nodes into one (Figure 1b). The second rule is *zero suppression*: If there exists a node $\mathtt{n}$ whose hi-child is $\bot$, we eliminate $\mathtt{n}$ and let all of the arcs pointed to $\mathtt{n}$ also point to $\mathsf{hi}(\mathtt{n})$ (Figure 1c). In the reduced ZDD, no node can be eliminated by applying the above two rules. Since applying these rules strictly decreases the size of ZDD, i.e., the number of nodes, we can deduce that the reduced ZDD of a family $\mathcal{F}$ is the smallest ZDD representing $\mathcal{F}$ given the total order $<$ of elements. The size of the reduced ZDD of the family $\mathcal{F}$, given the total order $<$, is denoted by $Z_<(\mathcal{F})$. If it is clear from the context, we omit $<$ and simply write it as $Z(\mathcal{F})$.

We briefly compare ZDDs with BDDs. BDD [2] has the same structure (syntax) as ZDD, although its semantics is slightly different. BDDs also follow the ordered property and have the smallest canonical form called *reduced BDD*. Given the total order $<$ of elements, the size of the reduced BDD of the family $\mathcal{F}$ is denoted by $B_<(\mathcal{F})$. The following is a famous result.

▶ **Lemma 2** ([13, Eq. (126)]). *For any family* $\mathcal{F}$ *of subsets of a set of* n *elements and any order* $<$ *of elements,* $B_<(\mathcal{F}) = O(nZ_<(\mathcal{F}))$ *and* $Z_<(\mathcal{F}) = O(nB_<(\mathcal{F}))$*.*

2.2 Family Algebra Operations on ZDDs

In this section, we explain how the family algebra operations are performed using ZDDs and point out what makes the difference between the basic set operations (union, intersection, difference, and symmetric difference) and the other operations.

As explained in Section 2.1, ZDD represents a family of sets in a recursive manner. Let us consider the situation in which there are two ZDDs whose root nodes are $\mathtt{n}$ and $\mathtt{m}$ and $\mathsf{lb}(\mathtt{n}) = \mathsf{lb}(\mathtt{m}) = x$. Then, the family of sets represented by them are $\mathcal{F}_{\mathtt{n}} = \mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{n})})$ and $\mathcal{F}_{\mathtt{m}} = \mathcal{F}_{\mathsf{lo}(\mathtt{m})} \cup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})$. The union of them is

$$\mathcal{F}_{\mathtt{n}} \cup \mathcal{F}_{\mathtt{m}} = [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [\{\{x\}\} \sqcup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \cup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})]. \tag{1}$$

This means that the ZDD representing $\mathcal{F}_{\mathtt{n}} \cup \mathcal{F}_{\mathtt{m}}$ can be described as follows: The root node's label is x, its lo-child represents $\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}$, and its hi-child represents $\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \cup \mathcal{F}_{\mathsf{hi}(\mathtt{m})}$. If $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathtt{m})$, we have a simpler recursion:

$$\mathcal{F}_{\mathtt{n}} \cup \mathcal{F}_{\mathtt{m}} = [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup \mathcal{F}_{\mathtt{m}}] \cup [\{\{\mathsf{lb}(\mathtt{n})\}\} \sqcup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \cup \mathcal{F}_{\mathtt{m}})]. \tag{2}$$

The case of $\mathsf{lb}(\mathtt{m}) < \mathsf{lb}(\mathtt{n})$ can be handled in the same way. By recursively expanding $\mathcal{F}_{\mathtt{n}} \cup \mathcal{F}_{\mathtt{m}}$ by (1) and (2), we eventually reach terminal nodes where the union is trivial, e.g., $\mathcal{F}_{\bot} \cup \mathcal{F}_{\top} = \{\emptyset\}$. Therefore, by caching the resultant ZDD nodes of $\mathcal{F}_{\mathtt{n}'} \cup \mathcal{F}_{\mathtt{m}'}$, where $\mathtt{n}'$ and $\mathtt{m}'$ are the child nodes of $\mathtt{n}$ and $\mathtt{m}$, respectively, we can efficiently compute the ZDD representing $\mathcal{F}_{\mathtt{n}} \cup \mathcal{F}_{\mathtt{m}}$. With the cache, one can show that we can build a ZDD representing the union of two ZDDs in a time proportional to the product of input ZDD sizes. The intersection, difference, and symmetric difference operations can be handled in almost the same way.

The other operations can also be performed in a recursive manner. However, the recursion becomes more complicated. Let us consider, for example, the join operation. When $\mathsf{lb}(\mathtt{n}) = \mathsf{lb}(\mathtt{m}) = x$, the join becomes

$$\begin{aligned}
\mathcal{F}_{\mathtt{n}} \sqcup \mathcal{F}_{\mathtt{m}} =& [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \cup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{n})})] \sqcup [\mathcal{F}_{\mathsf{lo}(\mathtt{m})} \cup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})] \\
=& [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})] \cup \\
& [(\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{n})}) \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [(\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{n})}) \sqcup (\{\{x\}\} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})] \\
=& [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [\{\{x\}\} \sqcup (\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})] \cup \\
& [\{\{x\}\} \sqcup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})})] \cup [\{\{x\}\} \sqcup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})] \\
=& [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [\{\{x\}\} \sqcup ((\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})}) \cup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}) \cup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})}))].
\end{aligned} \tag{3}$$

Here, the second equality holds because join distributes over the union. This means that we should build a ZDD where the root node's lo-child represents $\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}$ and its hi-child represents $(\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})}) \cup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{lo}(\mathtt{m})}) \cup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \sqcup \mathcal{F}_{\mathsf{hi}(\mathtt{m})})$. Thus, in the recursion, we should also compute the union $\cup$ of families, which also needs a recursion like that above. Another example is the restrict operation. Restrict can be computed as

$$\mathcal{F}_{\mathtt{n}} \vartriangle \mathcal{F}_{\mathtt{m}} = [\mathcal{F}_{\mathsf{lo}(\mathtt{n})} \vartriangle \mathcal{F}_{\mathsf{lo}(\mathtt{m})}] \cup [\{\{x\}\} \sqcup (\mathcal{F}_{\mathsf{hi}(\mathtt{n})} \vartriangle (\mathcal{F}_{\mathsf{lo}(\mathtt{m})} \cup \mathcal{F}_{\mathsf{hi}(\mathtt{m})}))]. \tag{4}$$

Thus, it is also necessary to compute the union of families as well as restrict.

Compared to the simple recursion for the computation of basic set operations, the complexity of such "double recursion" procedures are difficult to analyze.

3 Blow-Up Operations

3.1 High-Level Idea

As described in Section 2.2, the ZDD size after performing union or intersection can be bounded by the product of the sizes of operand ZDDs, i.e., $Z(\mathcal{F} \cup \mathcal{G}) = O(Z(\mathcal{F})Z(\mathcal{G}))$ and $Z(\mathcal{F} \cap \mathcal{G}) = O(Z(\mathcal{F})Z(\mathcal{G}))$. Thus, the ZDD of the union or intersection of *two* ZDDs remains polynomial-sized when the operand ZDDs have polynomial size. However, this does not hold for a non-constant number of ZDDs: even if $Z(\mathcal{F}_k) = O(\mathrm{poly}(m))$ for $k = 1, \ldots, m$, both $Z(\bigcup_{k=1}^{m} \mathcal{F}_k)$ and $Z(\bigcap_{k=1}^{m} \mathcal{F}_k)$ may become exponential in m.

We use such families to constitute examples of blow-up. More specifically, for each operation, we constitute an example such that performing this operation incurs the union or intersection of multiple families. Since we prove that the reduced ZDD representing the result of an operation will become exponential in size, we can confirm that *any* algorithm for computing the resultant ZDD incurs worst-case non-polynomial complexity. Combined with concrete instances, we prove that the worst-case complexity of family algebra operations is lower-bounded by an exponential factor.

We use the specific families of sets, hidden weighted bit function and permutation function, as explained below. Note that they are called “function” because they are originally defined as a Boolean function, but we here describe them as equivalent families of sets.

▶ **Definition 3.** *A* hidden weighted bit function $\mathcal{H}_m$ *is a family of sets defined as* $\{S \subseteq \{y_1, \ldots, y_m\} \mid y_{|S|} \in S\}$.

The hidden weighted bit function $\mathcal{H}_m$ can be represented as a union of elementary families. Define $\mathcal{E}_{m,k} \coloneqq \{S \subseteq \{y_1, \ldots, y_m\} \mid |S| = k, y_k \in S\}$, i.e., $\mathcal{E}_{m,k}$ consists of the subsets of $\{y_1, \ldots, y_m\}$ where the cardinality is k and y_k is contained. Then, $\mathcal{H}_m = \bigcup_{k=1}^{m} \mathcal{E}_{m,k}$. It can be easily verified that the size of the ZDD representing $Z(\mathcal{E}_{m,k})$ is $O(m^2)$ for any order of elements (see Section 3.4). However, it is known that the ZDD representing $\mathcal{H}_m$ must become exponential in size.

▶ **Theorem 4** ([3]). *For any order $<$ of elements, $B_<(\mathcal{H}_m) = \Omega(2^{m/5})$. Thus, by Lemma 2, $Z_<(\mathcal{H}_m) = \Omega(2^{m/5}/m)$.*

▶ **Definition 5.** *A* permutation function $\mathcal{P}_m$ *is a family of subsets of $\{y_1, \ldots, y_{m^2}\}$ such that (i) there is exactly one element from $y_{m(i-1)+1}, y_{m(i-1)+2}, \ldots, y_{m(i-1)+m}$ for $i = 1, \ldots, m$, and (ii) there is exactly one element from $y_j, y_{m+j}, \ldots, y_{m(m-1)+j}$ for $j = 1, \ldots, m$.*

The permutation function $\mathcal{P}_m$ is equivalent to the set of permutations: For $S \subseteq \{y_1, \ldots, y_{m^2}\}$, we associate a binary $m \times m$ matrix where the (i, j)-element is 1 if and only if $y_{m(i-1)+j} \in S$. Then, $S \in \mathcal{P}_m$ if and only if the associated matrix is a permutation matrix.

For $k = 1, \ldots, m$, let $\mathcal{Q}_{m,k}$ be the family of subsets of $\{y_1, \ldots, y_{m^2}\}$ such that there is exactly one element from $y_{m(k-1)+1}, y_{m(k-1)+2}, \ldots, y_{m(k-1)+m}$, and let $\mathcal{Q}_{m,m+k}$ be those such that there is exactly one element from $y_k, y_{m+k}, \ldots, y_{m(m-1)+k}$. Then, $\mathcal{P}_m = \bigcap_{k=1}^{2m} \mathcal{Q}_{m,k}$. Here, $Z(\mathcal{Q}_{m,k}) = O(m^2)$ for any order of elements, as proved in Section 3.4. However, it is again proved that the ZDD representing $\mathcal{P}_m$ must become exponential in size.

▶ **Theorem 6** ([13, Theorem K]). *For any order $<$ of elements, $B_<(\mathcal{P}_m) = \Omega(m2^m)$. Thus, by Lemma 2, $Z_<(\mathcal{P}_m) = \Omega(2^m/m)$.*

We first show the exponential blow-up cases for a specific order of elements in Section 3.2. However, we see that the size of ZDD representing the hidden weighted bit function or the permutation function is exponential regardless of the order of elements. Therefore, in Section 3.3, we prove that for each family generated by the operation in Section 3.2, the ZDD size representing it remains exponential regardless of the order of elements. This means that for each operation, there exists an instance in which the input ZDD size can be polynomial by managing the element order but the output ZDD size must be exponential for any order. Section 3.4 completes the proof by showing that some families can be represented by polynomial-sized ZDDs. Finally, Section 3.5 gives some discussions on the obtained result.

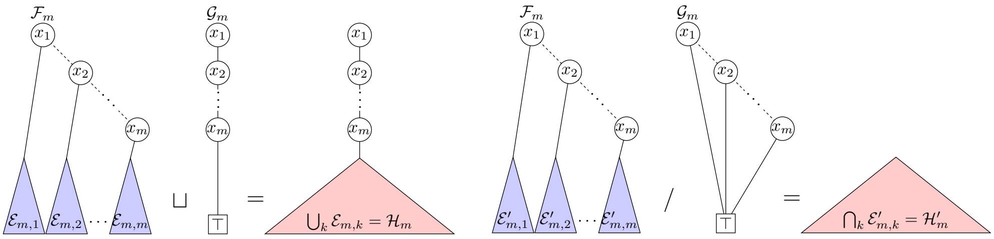


Figure 2 Example of blow-up for join (left) and quotient (right) operations. Blue triangles mean that the ZDD size representing this family is polynomial in m, while red triangle means that its size is exponential in m. Arcs going to $\bot$ terminal are omitted.

3.2 Proofs with Specific Element Order

3.2.1 Join, Disjoint Join, Joint Join, Meet, and Delta

For these operations, we constitute a pair of families that incur the union of $O(m)$ subfamilies. Combined with $\mathcal{E}_{m,k}$, the result after taking an operation contains $\bigcup_k \mathcal{E}_{m,k} = \mathcal{H}_m$, which is the hidden weighted bit function for which the ZDD size is exponential in m.

▶ **Theorem 7.** *Let $\diamond$ be a binary operator chosen from join ($\sqcup$), disjoint join ($\dot{\bowtie}$), joint join ($\hat{\bowtie}$), meet ($\sqcap$), and delta ($\boxplus$). Then, there exists a sequence of families $\mathcal{F}_m$ and $\mathcal{G}_m$ such that* (i) *$\mathcal{F}_m$ and $\mathcal{G}_m$ are families of subsets of a set of $O(m)$ elements,* (ii) *$Z(\mathcal{F}_m)+Z(\mathcal{G}_m) = O(m^3)$, and* (iii) *$Z(\mathcal{F}_m \diamond \mathcal{G}_m) = \Omega(2^{m/5}/m)$.*

Proof. Let us consider the families of subsets of $X \cup Y$, where $X := \{x_1, \dots, x_m\}$ and $Y := \{y_1, \dots, y_m\}$. We determine the order of elements as $x_1, \dots, x_m, y_1, \dots, y_m$. We define $\mathcal{F}_m$ as

$$\mathcal{F}_m := \bigcup_{k=1}^{m} (\{\{x_k\}\} \sqcup \mathcal{E}_{m,k}).$$

Since $Z(\mathcal{E}_{m,k}) = O(m^2)$ and the ZDD representing $\mathcal{F}_m$ becomes the left one of Figure 2 according to this order, $Z(\mathcal{F}_m) = O(m^3)$.

For the join operation, we let $\mathcal{G}_m := \{X\}$, where $Z(\mathcal{G}_m) = O(m)$. Then,

$$\begin{aligned}\mathcal{F}_m \sqcup \mathcal{G}_m &= (\textstyle\bigcup_{k=1}^{m}(\{\{x_k\}\} \sqcup \mathcal{E}_{m,k})) \sqcup \{X\} = \textstyle\bigcup_{k=1}^{m}((\{\{x_k\}\} \sqcup \mathcal{E}_{m,k}) \sqcup \{X\}) \\ &= \textstyle\bigcup_{k=1}^{m}(\{X\} \sqcup \mathcal{E}_{m,k}) = \{X\} \sqcup (\textstyle\bigcup_{k=1}^{m} \mathcal{E}_{m,k}) = \{X\} \sqcup \mathcal{H}_m,\end{aligned}$$

where the second and fourth equalities hold because join distributes over union and the third equality holds because $\{\{x_k\}\} \sqcup \{X\} = \{X\}$. Thus, the ZDD representing $\mathcal{F}_m \sqcup \mathcal{G}_m$ becomes the right one of Figure 2, meaning that the ZDD size is at least $Z(\mathcal{H}_m) = \Omega(2^{m/5}/m)$. Since every subset in $\mathcal{F}_m$ has at least one element from X, the result of joint join $\mathcal{F}_m \hat{\bowtie} \mathcal{G}_m$ also becomes $\{X\} \sqcup \mathcal{H}_m$, leading to an exponential-sized ZDD.

For the disjoint join operation, we let $\mathcal{G}_m := \bigcup_{k=1}^{m} \{X \setminus \{x_k\}\}$, where again $Z(\mathcal{G}_m) = O(m)$. Then, every subset in $\{\{x_k\}\} \sqcup \mathcal{E}_{m,k}$ has intersection with all of the subsets in $\mathcal{G}_m$, except for $X \setminus \{x_k\}$. Then,

$$\mathcal{F}_m \dot{\bowtie} \mathcal{G}_m = \textstyle\bigcup_{k=1}^{m}((\{x_k\} \cup (X \setminus \{x_k\})) \sqcup \mathcal{E}_{m,k}) = \{X\} \sqcup (\textstyle\bigcup_{k=1}^{m} \mathcal{E}_{m,k}) = \{X\} \sqcup \mathcal{H}_m,$$

meaning that $Z(\mathcal{F}_m \dot{\bowtie} \mathcal{G}_m) = \Omega(2^{m/5}/m)$.

For the meet operation, we let $\mathcal{G}_m := \{Y\}$, where $Z(\mathcal{G}_m) = O(m)$. Similar to join, we have $\mathcal{F}_m \sqcap \mathcal{G}_m = \mathcal{H}_m$, meaning that $Z(\mathcal{F}_m \sqcap \mathcal{G}_m) = \Omega(2^{m/5}/m)$.

For the delta operation, we let $\mathcal{G}_m = 2^X$. Since $\{\{x_k\}\} \boxplus 2^X = 2^X$ for any k, we have

$$\mathcal{F}_m \boxplus \mathcal{G}_m = \bigcup_{k=1}^{m}((\{\{x_k\}\} \boxplus 2^X) \sqcup \mathcal{E}_{m,k}) = 2^X \sqcup (\bigcup_{k=1}^{m} \mathcal{E}_{m,k}) = 2^X \sqcup \mathcal{H}_m.$$

The ZDD size of $\mathcal{F}_m \boxplus \mathcal{G}_m$ is at least $Z(\mathcal{H}_m) = \Omega(2^{m/5}/m)$. ◀

3.2.2 Quotient and Remainder

For the quotient operation, we constitute a pair of families such that performing an operation incurs the intersection of $O(m)$ subfamilies. Here, let $\mathcal{E}'_{m,k} := 2^Y \setminus \mathcal{E}_{m,k}$ be the complement of $\mathcal{E}_{m,k}$ regarding the family of subsets of Y. By De Morgan's laws, we have $\bigcap_k \mathcal{E}'_{m,k} = 2^Y \setminus (\bigcup_k \mathcal{E}_{m,k}) = 2^Y \setminus \mathcal{H}_m =: \mathcal{H}'_m$. The ZDD size representing $\mathcal{H}'_m$ can be lower bounded by the following lemma.

▶ **Lemma 8.** *Suppose that two families $\mathcal{F}, \mathcal{G}$ of subsets of the same set satisfy $Z(\mathcal{F}) = O(f(m))$, $Z(\mathcal{G}) = \Omega(g(m))$, and $\mathcal{F} \supseteq \mathcal{G}$. Then, $Z(\mathcal{F} \setminus \mathcal{G}) = \Omega(g(m)/f(m))$.*

Proof of Lemma 8. $\mathcal{F} \supseteq \mathcal{G}$ implies $\mathcal{F} \setminus (\mathcal{F} \setminus \mathcal{G}) = \mathcal{G}$. Since the ZDD size after taking the difference can be bounded by the product of the sizes of operand ZDDs, we have $Z(\mathcal{G}) = O(Z(\mathcal{F})Z(\mathcal{F} \setminus \mathcal{G}))$. Suppose $Z(\mathcal{F} \setminus \mathcal{G}) = o(g(m)/f(m))$. Then, $Z(\mathcal{G}) = o(f(m) \cdot (g(m)/f(m))) = o(g(m))$, refuting the assumption $Z(\mathcal{G}) = \Omega(g(m))$. Therefore, $Z(\mathcal{F} \setminus \mathcal{G}) = \Omega(g(m)/f(m))$. ◀

Since $Z(2^Y) = O(m)$ and $Z(\mathcal{H}_m) = \Omega(2^{m/5}/m)$, we have $Z(\mathcal{H}'_m) = \Omega(2^{m/5}/m^2)$.

▶ **Theorem 9.** *Let $\diamond$ be a binary operator chosen from quotient ($/$) and remainder ($\%$). Then, there exists a sequence of families $\mathcal{F}_m$ and $\mathcal{G}_m$ such that* (i) *$\mathcal{F}_m$ and $\mathcal{G}_m$ are families of subsets of a set of $O(m)$ elements,* (ii) *$Z(\mathcal{F}_m) + Z(\mathcal{G}_m) = O(m^3)$, and* (iii) *$Z(\mathcal{F}_m \diamond \mathcal{G}_m) = \Omega(2^{m/5}/\mathrm{poly}(m))$.*

Proof. We again consider the families of subsets of $X \cup Y$, where $X := \{x_1, \ldots, x_m\}$ and $Y := \{y_1, \ldots, y_m\}$. We use the same order of elements: $x_1, \ldots, x_m, y_1, \ldots, y_m$. We define $\mathcal{F}_m$ as

$$\mathcal{F}_m := \bigcup_{k=1}^{m}(\{\{x_k\}\} \sqcup \mathcal{E}'_{m,k}).$$

We have $Z(\mathcal{E}'_{m,k}) = O(m^2)$ as proved in Section 3.4, and thus $Z(\mathcal{F}_m) = O(m^3)$. We also define $\mathcal{G}_m := \{\{x_1\}, \ldots, \{x_m\}\}$, where $Z(\mathcal{G}_m) = O(m)$.

Let us consider $\mathcal{F}_m \,/\, \mathcal{G}_m$. By definition, $Y' \in \mathcal{F}_m \,/\, \mathcal{G}_m$ if and only if $Y' \subseteq Y$ and $\{x_k\} \cup Y' \in \mathcal{F}_m$ for $k = 1, \ldots, m$. From the definition of $\mathcal{F}_m$, it is equivalent to $Y' \in \bigcap_{k=1}^{m} \mathcal{E}'_{m,k}$. Thus, $\mathcal{F}_m \,/\, \mathcal{G}_m = \bigcap_{k=1}^{m} \mathcal{E}'_{m,k} = \mathcal{H}'_m$. This means $Z(\mathcal{F}_m \,/\, \mathcal{G}_m) = \Omega(2^{m/5}/m^2)$. The ZDDs involved are depicted in Figure 2.

For the remainder operation, we prepared the same families. Since $\mathcal{G}_m \sqcup (\mathcal{F}_m \,/\, \mathcal{G}_m) = \{\{x_1\}, \ldots, \{x_m\}\} \sqcup \mathcal{H}'_m$, $Z(\mathcal{G}_m \sqcup (\mathcal{F}_m \,/\, \mathcal{G}_m)) = \Omega(2^{m/5}/m^2)$. Also, since $S \in \mathcal{F}_m \,/\, \mathcal{G}_m$ if and only if $S \cup G \in \mathcal{F}_m$ for all $G \in \mathcal{G}_m$, all of the subsets in $\mathcal{G}_m \sqcup (\mathcal{F}_m \,/\, \mathcal{G}_m)$ are also contained in $\mathcal{F}_m$. In other words, $\mathcal{F}_m \supseteq \mathcal{G}_m \sqcup (\mathcal{F}_m \,/\, \mathcal{G}_m)$. Therefore, by using Lemma 8, $Z(\mathcal{F}_m \,\%\, \mathcal{G}_m) = \Omega((2^{m/5}/m^2)/m^3) = \Omega(2^{m/5}/m^5)$. ◀

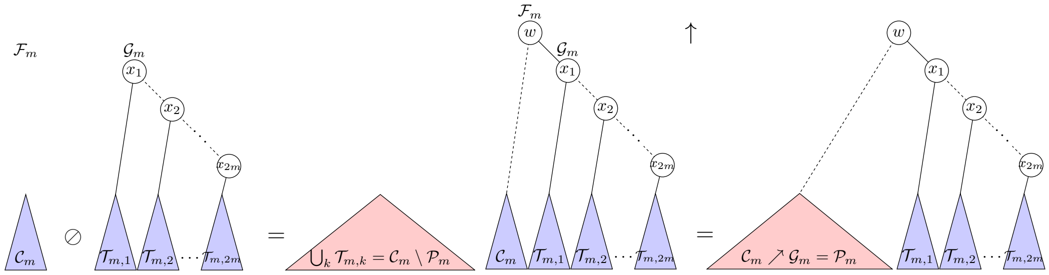


Figure 3 Example of blow-up for permit (left) and maximal (right) operations.

3.2.3 Restrict, Permit, Nonsuperset, and Nonsubset

These operations include inclusion relations of subsets in their definitions, which makes it difficult to generate a hidden weighted bit function as a result of the operation. This is due to the fact that $\mathcal{H}_m$ includes the universal set Y as well as a singleton $\{y_1\}$. For example, if $\mathcal{F}$ is the family of subsets of Y and the universal set Y is included in the result of $\mathcal{F} \oslash \mathcal{G}$, all of the subsets in $\mathcal{F}$ must be included in $\mathcal{F} \oslash \mathcal{G}$ due to the definition of the permit operation.

Instead, we use the permutation function. Because every set in $\mathcal{P}_m$ has cardinality m, the above issue can be alleviated. More specifically, we prepared the complement of the families:

$$\mathcal{C}_m \coloneqq \{S \subseteq \{y_1, \dots, y_{m^2}\} \mid |S| = m\}, \quad \mathcal{T}_{m,k} \coloneqq \mathcal{C}_m \setminus \mathcal{Q}_{m,k} (= \mathcal{C}_m \cap (2^Y \setminus \mathcal{Q}_{m,k})).$$

Here, $\mathcal{C}_m$ is the family of subsets with cardinality m, and thus $\mathcal{T}_{m,k}$ also contains only the subsets with cardinality m. Moreover, by De Morgan's laws,

$$\bigcup_{k=1}^{2m} \mathcal{T}_{m,k} = \mathcal{C}_m \cap \left(\bigcup_{k=1}^{2m} (2^Y \setminus \mathcal{Q}_{m,k}) \right) = \mathcal{C}_m \cap \left(2^Y \setminus \left(\bigcap_{k=1}^{2m} \mathcal{Q}_{m,k} \right) \right) = \mathcal{C}_m \setminus \mathcal{P}_m.$$

We use these families $\mathcal{T}_{m,k}$ to prove the following.

▶ **Theorem 10.** *Let $\diamond$ be a binary operator chosen from restrict ($\triangle$), permit ($\oslash$), nonsuperset ($\searrow$), and nonsubset ($\nearrow$). Then, there exists a sequence of families $\mathcal{F}_m$ and $\mathcal{G}_m$ such that* (i) *$\mathcal{F}_m$ and $\mathcal{G}_m$ are families of subsets of a set of $O(m^2)$ elements,* (ii) *$Z(\mathcal{F}_m) + Z(\mathcal{G}_m) = O(m^4)$, and* (iii) *$Z(\mathcal{F}_m \diamond \mathcal{G}_m) = \Omega(2^m/\mathrm{poly}(m))$.*

Proof. Let us consider the families of subsets of $X \cup Y$, where $X \coloneqq \{x_1, \dots, x_{2m}\}$ and $Y \coloneqq \{y_1, \dots, y_{m^2}\}$. The order of elements is $x_1, \dots, x_{2m}$ followed by $y_1, \dots, y_{m^2}$.

We first consider the permit operation. We define $\mathcal{F}_m \coloneqq \mathcal{C}_m$ and

$$\mathcal{G}_m \coloneqq \bigcup_{k=1}^{2m} (\{\{x_k\}\} \sqcup \mathcal{T}_{m,k}).$$

As proved in Section 3.4, $Z(\mathcal{C}_m) = O(m^3)$ and $Z(\mathcal{T}_{m,k}) = O(m^3)$. Thus, $Z(\mathcal{F}_m) = O(m^3)$ and $Z(\mathcal{G}_m) = O(m^4)$. Any set in $\mathcal{F}_m = \mathcal{C}_m$ consists of m elements chosen from $y_1, \dots, y_{m^2}$, and any set in $\mathcal{G}_m$ consists of m elements from $y_1, \dots, y_{m^2}$ plus one element from $x_1, \dots, x_{2m}$. Thus, set $S \in \mathcal{F}_m$ is a subset of some set in $\mathcal{G}_m$ if and only if $\{x_k\} \cup S \in \mathcal{G}_m$ for some k. In other words, $S \in \mathcal{F}_m \oslash \mathcal{G}_m$ if and only if S is included in $\mathcal{T}_{m,k}$ for some k. Since $\mathcal{C}_m \supset \mathcal{T}_{m,k}$ for any k by definition, this means $\mathcal{F}_m \oslash \mathcal{G}_m = \bigcup_{k=1}^{2m} \mathcal{T}_{m,k} = \mathcal{C}_m \setminus \mathcal{P}_m$. Since $Z(\mathcal{C}_m) = O(m^3)$ and $Z(\mathcal{P}_m) = \Omega(2^m/m)$, we have $Z(\mathcal{F}_m \oslash \mathcal{G}_m) = \Omega(2^m/m^4)$ by Lemma 8. The ZDDs involved are depicted in Figure 3.

The nonsubset operation can be treated with the same families. Since $\mathcal{F}_m \nearrow \mathcal{G}_m = \mathcal{F}_m \setminus (\mathcal{F}_m \oslash \mathcal{G}_m)$ by definition, we have $\mathcal{F}_m \nearrow \mathcal{G}_m = \mathcal{C}_m \setminus (\mathcal{C}_m \setminus \mathcal{P}_m) = \mathcal{P}_m$, where the last equality holds due to $\mathcal{C}_m \supset \mathcal{P}_m$. Thus, $Z(\mathcal{F}_m \nearrow \mathcal{G}_m) = \Omega(2^m/m)$.

The restrict and nonsuperset operations can be handled by nearly the same families. We define the same $\mathcal{G}_m$ and let $\mathcal{F}_m \coloneqq \{X\} \sqcup \mathcal{C}_m$. Similar to the proof of the permit operation, set $X \cup S \in \mathcal{F}_m$ $(S \subseteq Y)$ is a superset of some sets in $\mathcal{G}_m$ if and only if $\{x_k\} \cup S \in \mathcal{G}_m$ for some k. This means $\mathcal{F}_m \vartriangle \mathcal{G}_m = \{X\} \sqcup (\bigcup_{k=1}^{2m} \mathcal{T}_{m,k}) = \{X\} \sqcup (\mathcal{C}_m \setminus \mathcal{P}_m)$, whose ZDD size is $\Omega(2^m/m^4)$. For the nonsuperset operation, we have $\mathcal{F}_m \searrow \mathcal{G}_m = \mathcal{F}_m \setminus (\mathcal{F}_m \vartriangle \mathcal{G}_m) = \{X\} \sqcup \mathcal{P}_m$, yielding $Z(\mathcal{F}_m \searrow \mathcal{G}_m) = \Omega(2^m/m)$. ◀

3.2.4 Maximal and Minimal

For these operations, we use the close relationship with the nonsuperset and nonsubset operations. We prepare a family having $\mathcal{F}_m$ and $\mathcal{G}_m$ appearing in the proof of Theorem 10 as a subfamily.

▶ **Theorem 11.** *Let $^{\diamond}$ be a unary operator chosen from maximal $(^{\uparrow})$ and minimal $(^{\downarrow})$. Then, there exists a sequence of families $\mathcal{F}_m$ such that* (i) *$\mathcal{F}_m$ is a family of subsets of a set of $O(m^2)$ elements,* (ii) *$Z(\mathcal{F}_m) = O(m^4)$, and* (iii) *$Z(\mathcal{F}_m^{\diamond}) = \Omega(2^m/\mathrm{poly}(m))$.*

Proof. Let us consider the family of subsets of $\{w\} \cup X \cup Y$, where $X \coloneqq \{x_1, \ldots, x_{2m}\}$ and $Y \coloneqq \{y_1, \ldots, y_{m^2}\}$. The order of elements is $w, x_1, \ldots, x_{2m}$ followed by $y_1, \ldots, y_{m^2}$.

We first consider the maximal operation. We define $\mathcal{F}_m$ as

$$\mathcal{F}_m \coloneqq \mathcal{C}_m \cup [\{\{w\}\} \sqcup \mathcal{G}_m], \quad \text{where } \mathcal{G}_m \coloneqq \bigcup_{k=1}^{2m} (\{\{x_k\}\} \sqcup \mathcal{T}_{m,k}).$$

Here, we observe that this $\mathcal{G}_m$ is the same as that appearing in the proof of Theorem 10. The ZDD size is bounded as $Z(\mathcal{F}_m) = O(Z(\mathcal{C}_m) + Z(\mathcal{G}_m)) = O(m^4)$. Every set in $\mathcal{C}_m$ has m elements and every set in $\{\{w\}\} \sqcup \mathcal{G}_m$ has $m+2$ elements. Thus, every set in the latter family is maximal, while a set in the former family is maximal if and only if it is not a subset of any set included in the latter family. Therefore, we have

$$\mathcal{F}_m^{\uparrow} = [\mathcal{C}_m \nearrow (\{\{w\}\} \sqcup \mathcal{G}_m)] \cup [\{\{w\}\} \sqcup \mathcal{G}_m] = [\mathcal{C}_m \nearrow \mathcal{G}_m] \cup [\{\{w\}\} \sqcup \mathcal{G}_m] = \mathcal{P}_m \cup [\{\{w\}\} \sqcup \mathcal{G}_m],$$

where the second equality holds because all of the sets in $\mathcal{C}_m$ do not include w and the last equality follows from the proof of Theorem 10. The resultant ZDD is like the right one in Figure 3, which implies $Z(\mathcal{F}_m^{\uparrow}) \geq Z(\mathcal{P}_m) = \Omega(2^m/m)$.

The minimal can be treated in a similar way. We define

$$\mathcal{F}_m \coloneqq \mathcal{G}_m \cup [\{\{w\}\} \sqcup \{\{x_1, \ldots, x_{2m}\}\} \sqcup \mathcal{C}_m],$$

where $\mathcal{G}_m$ is the same family as that above. We again have $Z(\mathcal{F}_m) = O(m^4)$. Every set in $\mathcal{G}_m$ has $m+1$ elements and every set in $\{\{w\}\} \sqcup \{X\} \sqcup \mathcal{C}_m$ has $3m+1$ elements. Thus, every set in the former family is minimal, while a set in the latter family is minimal if and only if it is not a superset of any set included in the former family. Now we have

$$\begin{aligned} \mathcal{F}_m^{\downarrow} &= \mathcal{G}_m \cup [(\{\{w\}\} \sqcup \{X\} \sqcup \mathcal{C}_m) \searrow \mathcal{G}_m] \\ &= \mathcal{G}_m \cup [\{\{w\}\} \sqcup ((\{X\} \sqcup \mathcal{C}_m) \searrow \mathcal{G}_m)] = \mathcal{G}_m \cup [\{\{w\}\} \sqcup \{X\} \sqcup \mathcal{P}_m], \end{aligned}$$

where the second equality holds because none of the sets in $\mathcal{G}_m$ includes w and the last equality follows from the proof of Theorem 10. This again implies $Z(\mathcal{F}_m^{\downarrow}) \geq Z(\mathcal{P}_m) = \Omega(2^m/m)$. ◀

3.2.5 Minimal Hitting Set and Closure

For these operations, we can constitute much simpler examples.

▶ **Theorem 12.** *Let* $^\diamond$ *be a unary operator chosen from minimal hitting set* ($^\sharp$) *and closure* ($^\cap$). *Then, there exists a sequence of families* $\mathcal{F}_m$ *such that* (i) $\mathcal{F}_m$ *is a family of subsets of a set of* $O(m^2)$ *elements,* (ii) $Z(\mathcal{F}_m) = O(m^4)$*, and* (iii) $Z(\mathcal{F}_m^\diamond) = \Omega(2^m/\mathrm{poly}(m))$.

Proof. Let $X \coloneqq \{x_1, \ldots, x_{2m}\}$ and $Y \coloneqq \{y_1, \ldots, y_{m^2}\}$. For $k = 1, \ldots, m$, we set $S_k \coloneqq \{y_{m(k-1)+1}, y_{m(k-1)+2}, \ldots, y_{m(k-1)+m}\}$, and $S_{m+k} \coloneqq \{y_k, y_{m+k}, \ldots, y_{m(m-1)+k}\}$.

For minimal hitting set operation, we consider a family of subsets of Y. We define $\mathcal{F}_m \coloneqq \{S_1, \ldots, S_{2m}\}$. Since the ZDD size can be bounded by the sum of cardinality of a set in the family [16], $Z(\mathcal{F}_m) \leq \sum_i |S_i| = O(m^2)$. For $S \subseteq \{y_1, \ldots, y_{m^2}\}$, we associate a binary $m \times m$ matrix, where the (i,j)-element is 1 if and only if $y_{m(i-1)+j} \in S$. Then, $S \cap S_k \neq \emptyset$ means that the k-th row of the matrix has at least one 1 and $S \cap S_{m+k} \neq \emptyset$ means that the k-th column of the matrix has at least one 1. Thus, $S \in \mathcal{F}_m^\sharp$ if and only if the corresponding matrix has at least one 1 for any column or row and no proper subset of S satisfies this property. The minimal matrix having this property is the permutation matrix, and thus $\mathcal{F}_m^\sharp = \mathcal{P}_m$, that is, the permutation function. This implies $Z(\mathcal{F}_m^\sharp) = \Omega(2^m/m)$.

For closure operation, we consider a family of subsets of $X \cup Y$. For $k = 1, \ldots, m$ and $\ell = 1, \ldots, m$, we define $R_{k,\ell} \coloneqq (X \setminus \{x_k, x_{m+\ell}\}) \cup ((Y \setminus S_k \setminus S_{m+\ell}) \cup \{y_{m(k-1)+\ell}\})$. We define $\mathcal{F}_m \coloneqq \{R_{k,\ell} \mid k, \ell = 1, \ldots, m\}$. Again, since the ZDD size can be bounded by the sum of cardinality of a set in the family [16], $Z(\mathcal{F}_m) \leq \sum_{k,\ell} |R_{k,\ell}| = O(m^4)$. Then, we show that $\mathcal{F}_m^\cap \cap \mathcal{C}_m = \mathcal{P}_m$, where $\mathcal{P}_m$ is the permutation function. If it is shown, $Z(\mathcal{F}_m^\cap \cap \mathcal{C}_m) = Z(\mathcal{P}_m) = \Omega(2^m/m)$. On the other hand, $Z(\mathcal{F}_m^\cap \cap \mathcal{C}_m) = O(Z(\mathcal{F}_m^\cap) Z(\mathcal{C}_m))$. Since $Z(\mathcal{C}_m) = O(m^3)$, we can deduce that $Z(\mathcal{F}_m^\cap) = \Omega(2^m/\mathrm{poly}(m))$.

We now prove $\mathcal{F}_m^\cap \cap \mathcal{C}_m = \mathcal{P}_m$. First, we show that $\mathcal{F}_m^\cap \cap \mathcal{C}_m \subseteq \mathcal{P}_m$. $R_{k,\ell}$ does not contain any element in S_k and $S_{m+\ell}$ except for $y_{m(k-1)+\ell}$. By fixing k, if $\mathcal{F}' \subseteq \mathcal{F}$ contains at least one $R_{k,\ell}$ for some ℓ, $S = \bigcap_{S' \in \mathcal{F}'} S'$ contains at most one element from S_k. Moreover, S does not contain x_k if and only if $\mathcal{F}'$ contains at least one $R_{k,\ell}$ for some ℓ. Similarly, by fixing ℓ, if $\mathcal{F}'$ contains at least one $R_{k,\ell}$ for some k, which is equivalent to that S does not contain $x_{m+\ell}$, S contains at most one element from $S_{m+\ell}$. Now we can say that when S contains no element in X, S contains at most one element in S_k for any $k = 1, \ldots, 2m$. This means that if S contains no element in X and m elements in Y, $S \in \mathcal{P}_m$. Thus, $\mathcal{F}_m^\cap \cap \mathcal{C}_m \subseteq \mathcal{P}_m$. Next, we show that $\mathcal{F}_m^\cap \cap \mathcal{C}_m \supseteq \mathcal{P}_m$. Let σ be an arbitrary permutation of $1, \ldots, m$. Then, $\{y_{\sigma(1)}, y_{m+\sigma(2)}, \ldots, y_{(m-1)m+\sigma(m)}\} = R_{1,\sigma(1)} \cap R_{2,\sigma(2)} \cap \cdots \cap R_{m,\sigma(m)}$. This means that any set in $\mathcal{P}_m$ is in $\mathcal{F}_m^\cap$. Thus, $\mathcal{F}_m^\cap \cap \mathcal{C}_m \supseteq \mathcal{P}_m$. This concludes $\mathcal{F}_m^\cap \cap \mathcal{C}_m = \mathcal{P}_m$. ◀

3.3 Consideration for Element Order

The above proofs fix the order of elements for each operation. Thus, there is still a possibility that the resultant ZDD size becomes smaller by managing the order of elements. However, it seems that the size of resultant ZDD remains exponential regardless of the order of elements, since every resultant family contains a hidden weighted bit function, a permutation function, or similar families as a subfamily. In the following, we prove that every resultant family has an exponential ZDD size regardless of the order of elements.

▶ **Definition 13.** *Let* $\mathcal{F}$ *be a family of subsets of set* X*, and let* Y, Y' *be the subsets of* X *satisfying* $Y \cap Y' = \emptyset$. *We define* $\mathcal{F}|_{Y,Y'}$ *as the family of subsets of* $X \setminus (Y \cup Y')$ *such that* $S \in \mathcal{F}|_{Y,Y'}$ *if and only if* $S \cup Y \in \mathcal{F}$.

In other words, $\mathcal{F}|_{Y,Y'}$ is the family of sets generated from $\mathcal{F}$ by first extracting the sets containing every element of Y, but no element of Y', and then eliminating all of the elements of Y from every set. This operation is called *conditioning* and it is a famous result that this can be performed in polynomial time with BDDs [6]. For the sake of completeness, we show this can also be performed in polynomial time with ZDDs, and then we prove the following.

▶ **Lemma 14.** *Let $\mathcal{F}$ be a family of subsets of a set X of $O(f(m))$ elements. If there exist $Y, Y' \subseteq X$ such that $Z_<(\mathcal{F}|_{Y,Y'}) = \Omega(g(m))$ for any order $<$ of elements, we have $Z_<(\mathcal{F}) = \Omega(g(m)/f(m))$ for any order $<$ of elements.*

If this lemma holds, we can show that the resultant families in Section 3.2 all have an exponential ZDD size regardless of the order of elements. This is because the resultant families in Section 3.2 all have a hidden weighted bit function, a permutation function, or its complements as a subfamily and all of them have an exponential ZDD size regardless of the order of elements; a detailed discussion is given later.

Proof of Lemma 14. If we can show $Z_<(\mathcal{F}|_{Y,Y'}) = O(Z_<(\mathcal{F})f(m))$ for any $Y, Y' \subseteq X$ and any order $<$ of elements, Lemma 14 can be proved as follows: Suppose that there is an order $<$ of elements satisfying $Z_<(\mathcal{F}) = o(g(m)/f(m))$. Then, by the above equation, we have $Z_<(\mathcal{F}|_{Y,Y'}) = o((g(m)/f(m)) \cdot f(m)) = o(g(m))$. This contradicts the assumption that $Z_<(\mathcal{F}|_{Y,Y'}) = \Omega(g(m))$ for any order $<$ of elements.

Next, we fix an arbitrary order $<$ of elements and show $Z_<(\mathcal{F}|_{Y,Y'}) = O(Z_<(\mathcal{F})f(m))$. Here, we consider the operations for constructing a ZDD representing $\mathcal{F}|_{Y,Y'}$ from the ZDD of $\mathcal{F}$. We first extract the sets that contain every element of Y but do not contain any element of Y'. Then, we eliminate all elements of Y.

The former step can be achieved by the intersection operation. Let $\mathcal{G}$ be the family of subsets of X such that $S \in \mathcal{G}$ if and only if S contains all of the elements in Y but does not contain any element in Y'. In other words, $\mathcal{G} := \{S \subseteq X \mid S \cap Y = Y \wedge S \cap Y' = \emptyset\}$. Then, $\mathcal{F} \cap \mathcal{G}$ is the desired family. The ZDD representing $\mathcal{G}$ has the following form: (i) For any $x \in Y$, there is only one ZDD node labeled x whose lo-child is $\bot$ while its hi-child is the next-level node. (ii) For any $x \in Y'$, there is no node labeled x by the reduction rule of ZDD. (iii) for any $x \in X \setminus (Y \cup Y')$, there is only one ZDD node labeled x whose lo-child and hi-child are both the next-level node. Thus, we have $Z_<(\mathcal{G}) = O(f(m))$ because the base set X of $\mathcal{F}$ has $O(f(m))$ elements and, for any element $x \in X$, there is at most one node labeled x. Finally, $Z_<(\mathcal{F} \cap \mathcal{G}) = O(Z_<(\mathcal{F})f(m))$.

The latter can be achieved by eliminating the nodes labeled $x \in Y$ and replacing the branches heading it. For a node labeled $x \in Y$, its lo-child must be $\bot$, since the ZDD is reduced and every set in $\mathcal{F} \cap \mathcal{G}$ must contain x. For this node, we first make all of the arcs heading to it point to its hi-child. Then, we eliminate this node. By performing this operation for every node labeled $x \in Y$, we finally obtain the ZDD of $\mathcal{F}|_{Y,Y'}$. Since this operation does not increase the size of ZDD, we have $Z_<(\mathcal{F}|_{Y,Y'}) = O(Z_<(\mathcal{F})f(m))$. ◀

Now we can show that the resultant families in the proof of Section 3.2 have exponential ZDD size regardless of the order of elements. For example, for the join operation, $(\{X\} \sqcup \mathcal{H}_m)|_{X,\emptyset} = \mathcal{H}_m$ and $Z(\mathcal{H}_m) = \Omega(2^{m/5}/m)$ for any order of elements of Y (and thus that of $X \cup Y$). Therefore, by Lemma 14, $Z(\mathcal{F}_m \sqcup \mathcal{G}_m) = \Omega(2^{m/5}/m^2)$ for any order of elements of $X \cup Y$. Similar arguments hold for the other operations. We here show that all the resultant families in the proof of Section 3.2 have exponential ZDD size regardless of the order of elements.

Disjoint join $\dot{\bowtie}$ and joint join $\hat{\bowtie}$: The resultant family of these operations in the proof of Theorem 7 is $(\{X\} \sqcup \mathcal{H}_m)$. Here, $(\{X\} \sqcup \mathcal{H}_m)|_{X,\emptyset} = \mathcal{H}_m$.

Meet $\sqcap$: In the proof of Theorem 7, we already have $\mathcal{F}_m \sqcap \mathcal{G}_m = \mathcal{H}_m$. Thus, $Z(\mathcal{F}_m \sqcap \mathcal{G}_m) = \Omega(2^{m/5}/m)$ for any order of elements.

Delta $\boxplus$: In the proof of Theorem 7, we have $\mathcal{F}_m \boxplus \mathcal{G}_m = 2^X \sqcup \mathcal{H}_m$. Since $(2^X \sqcup \mathcal{H}_m)|_{X,\emptyset} = \mathcal{H}_m$, $Z(\mathcal{F}_m \boxplus \mathcal{G}_m) = \Omega(2^{m/5}/\mathrm{poly}(m))$ for any order of elements.

Quotient $/$: $Z(2^Y) = O(m)$ and $Z(\mathcal{H}_m) = \Omega(2^{m/5}/m)$ for any order of elements, and $Z(\mathcal{H}'_m) = \Omega(2^{m/5}/m^2)$ for any order of elements by Lemma 8. This also holds for $\mathcal{F}_m \,/\, \mathcal{G}_m$ in the proof of Theorem 9 since it equals $\mathcal{H}'_m$.

Remainder $\%$: Since $\mathcal{F}_m = \bigcup_k(\{\{x_k\}\} \sqcup \mathcal{E}'_{m,k})$ and $\mathcal{G}_m \sqcup (\mathcal{F}_m \,/\, \mathcal{G}_m) = \{\{x_1\}, \ldots, \{x_m\}\} \sqcup \mathcal{H}'_m$, $\mathcal{F}_m \,\%\, \mathcal{G}_m = \bigcup_k(\{\{x_k\}\} \sqcup (\mathcal{E}'_{m,k} \setminus \mathcal{H}'_m))$. Thus, $(\mathcal{F}_m \,\%\, \mathcal{G}_m)|_{\{x_1\},X\setminus\{x_1\}} = \mathcal{E}'_{m,1} \setminus \mathcal{H}'_m$. Here, $Z(\mathcal{E}'_{m,1}) = O(m^2)$ and $Z(\mathcal{H}'_m) = \Omega(2^{m/5}/m^2)$ for any order of elements, and $Z(\mathcal{E}'_{m,1} \setminus \mathcal{H}'_m) = \Omega(2^{m/5}/m^4)$ for any order of elements by Lemma 8. Thus, by Lemma 14, $Z(\mathcal{F}_m \,\%\, \mathcal{G}_m) = \Omega(2^{m/5}/m^6)$ for any order of elements because it is a family of subsets of a set with $O(m^2)$ elements.

Permit $\oslash$ and nonsubset $\nearrow$: We already have $\mathcal{F}_m \oslash \mathcal{G}_m = \mathcal{C}_m \setminus \mathcal{P}_m$ and $\mathcal{F}_m \nearrow \mathcal{G}_m = \mathcal{P}_m$ in the proof of Theorem 10. Since $Z(\mathcal{C}_m) = O(m^3)$ and $\mathcal{P}_m = \Omega(2^m/m)$ for any order of elements, $Z(\mathcal{C}_m \setminus \mathcal{P}_m) = \Omega(2^m/m^4)$ for any order of elements by Lemma 8.

Restrict $\triangle$ and nonsuperset $\searrow$: We have $(\{X\} \sqcup (\mathcal{C}_m \setminus \mathcal{P}_m))|_{X,\emptyset} = \mathcal{C}_m \setminus \mathcal{P}_m$ and $(\{X\} \sqcup \mathcal{P}_m)|_{X,\emptyset} = \mathcal{P}_m$; see the proof of Theorem 10.

Maximal $^{\uparrow}$: In the proof of Theorem 11, we have $\mathcal{F}_m^{\uparrow}|_{\emptyset,\{w\}\cup X} = \mathcal{P}_m$.

Minimal $^{\downarrow}$: In the proof of Theorem 11, we have $\mathcal{F}_m^{\downarrow}|_{\{w\}\cup X,\emptyset} = \mathcal{P}_m$.

Minimal hitting set $^{\sharp}$: In the proof of Theorem 12, we already have $\mathcal{F}_m^{\sharp} = \mathcal{P}_m$.

Closure $^{\cap}$: In the proof of Theorem 12, we have $\mathcal{F}_m^{\cap} \cap \mathcal{C}_m = \mathcal{P}_m$. Since $Z(\mathcal{C}_m) = O(m^3)$ for any order of elements, $Z(\mathcal{F}_m^{\cap}) = \Omega(2^m/\mathrm{poly}(m))$ for any order of elements.

3.4 Polynomially Bounded ZDDs

We complete the proof of this section by showing that the ZDD sizes of some families appearing in the previous proofs are bounded by a polynomial of m. To prove the size bound, we consider the following *linear network model* to distinguish whether a set is contained in the family $\mathcal{F}$. Note that the idea of a linear network model comes from Knuth's book [13, Theorem M], where it was used to prove the bound of BDD size. Suppose that the order of elements is $x_1 < x_2 < \cdots < x_n$. There are n computational modules $M_1, \ldots, M_n$. Module M_i receives an input of one bit indicating whether x_i is included in the set. Module M_i sends a_{i+1} bits of information to module M_{i+1}. Overall, every module M_i receives an input x_i and a_i bits of information from M_{i-1} and sends a_{i+1} bits of information to M_{i+1}. Since module M_1 has no preceding module, we set $a_1 = 0$. The final module, M_n, outputs one bit indicating whether the set is included in the family $\mathcal{F}$. An overview of the linear network model is drawn in Figure 4. The following lemma suggests that if we can construct a small linear network for the family $\mathcal{F}$, the ZDD size of $\mathcal{F}$ can be bounded.

▶ **Lemma 15.** *For family $\mathcal{F}$ of subsets of $\{x_1, \ldots, x_n\}$, assume that we can construct the linear network model described above to distinguish whether a set is contained in $\mathcal{F}$. Then, the size of ZDD representing $\mathcal{F}$ is bounded by $Z(\mathcal{F}) \le 2 + \sum_{i=1}^{n} 2^{a_i}$.*

Proof. For $k = 1, \ldots, n$, we consider the number of distinct subfamilies $\mathcal{F}|_{X,Y}$, where $X \cup Y = \{x_1, \ldots, x_{k-1}\}$. This is because by the node sharing rule, the number of nodes labeled x_k is upper-bounded by the number of possible distinct subfamilies.

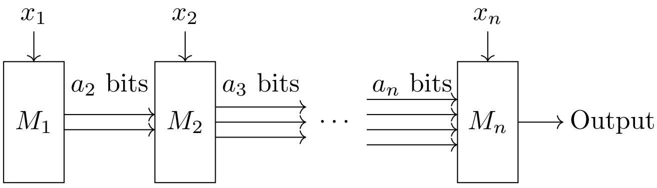


Figure 4 Schematic overview of linear network model.

We observe that the input to module M_i is a_i bits. This means that, regardless of the inclusion of $x_1, \dots, x_{k-1}$, the subfamily $\mathcal{F}|_{X,Y}$ is completely determined by the information of a_i bits. Therefore, there are at most 2^{a_i} distinct subfamilies, yielding the result that the number of nodes labeled x_k is upper-bounded by 2^{a_i}. Since there are two terminal nodes $\top$ and $\bot$, the overall ZDD size is bounded by $Z(\mathcal{F}) \le 2 + \sum_{i=1}^{n} 2^{a_i}$. ◀

By Lemma 15, we only have to consider a small linear network for every family.

$\mathcal{E}_{m,k}$ in Section 3.1: The family $\mathcal{E}_{m,k}$ is defined as $\{S \subseteq \{y_1, \dots, y_m\} \mid |S| = k, y_k \in S\}$. In judging whether $S \in \mathcal{E}_{m,k}$ with a linear network, the module M_t is only concerned with the number of elements from $y_1, \dots, y_t$ in S and whether y_k is in S. The former information can be represented with $\lceil \log(m+1) \rceil$ bits and the latter can be represented with 1 bit. Thus, we can construct a linear network with $a_t = \lceil \log(m+1) \rceil + 1$ bits. By Lemma 15, we have $Z(\mathcal{E}_{m,k}) \le 2 + m2^{\lceil \log(m+1) \rceil + 1} = O(m^2)$.

$\mathcal{Q}_{m,k}$ in Section 3.1: Each of the families $\mathcal{Q}_{m,k}$ $(k = 1, \dots, 2m)$ is the family of subsets of $\{y_1, \dots, y_{m^2}\}$ such that there is exactly one element from a set of m selected elements. In constructing a linear network, the module M_t is only concerned with the number of selected elements in S: zero, one, or more than one. This information can be represented with 2 bits. Thus, we have $Z(\mathcal{Q}_{m,k}) \le 2 + m^2 2^2 = O(m^2)$.

$\mathcal{E}'_{m,k}$ in Section 3.2.2: The linear network for $\mathcal{E}'_{m,k} = 2^Y \setminus \mathcal{E}_{m,k}$ can be the same as that for $\mathcal{E}_{m,k}$, except that the output is inverted. Thus, $Z(\mathcal{E}'_{m,k}) = O(m^2)$.

$\mathcal{C}_m$ in Section 3.2.3: The family $\mathcal{C}_m$ is defined as $\{S \subseteq \{y_1, \dots, y_{m^2}\} \mid |S| = m\}$. Similar to the case of $\mathcal{Q}_{m,k}$, every module only retains the number of elements from $y_1, \dots, y_t$ in S. Moreover, we should only count this number until m; if the count exceeds m, we can immediately determine that S is not in $\mathcal{C}_m$. This count value can be represented with $\lceil \log(m+2) \rceil$ bits. Thus, we have $Z(\mathcal{C}_m) \le 2 + m^2 2^{\lceil \log(m+2) \rceil} = O(m^3)$.

$\mathcal{T}_{m,k}$ in Section 3.2.3: For $\mathcal{T}_{m,k} = \mathcal{C}_m \setminus \mathcal{Q}_{m,k}$, we can construct a linear network by combining the networks for $\mathcal{C}_m$ and $\mathcal{Q}_{m,k}$. We have $\lceil \log(m+2) \rceil$ bits for $\mathcal{C}_m$ and 2 bits for $\mathcal{Q}_{m,k}$. Thus, we have $Z(\mathcal{T}_{m,k}) \le 2 + m^2 2^{\lceil \log(m+2) \rceil + 2} = O(m^3)$.

We finally note that the ZDD sizes of the above families remain polynomial in m even if the order of elements is different from $y_1 < y_2 < \cdots < y_m < \cdots < y_{m^2}$. Since the cardinality constraint is symmetric, we can reuse the same linear network for different orders of elements. The existence of specific elements can also be treated by changing the input that is watched.

3.5 Discussion

Finally, we give some discussions for the presented results. First, we argue theoretical results for BDDs. As stated in Lemma 2, the sizes of BDD and ZDD differ only by a linear factor of the size of the base item set. All the results in Section 3.2 have the same form that the number of elements is $O(\text{poly}(m))$, the input ZDD sizes are $O(\text{poly}(m))$, and the output ZDD size is exponential in m. Therefore, even if these families are represented by BDDs, the input BDD sizes are all $O(\text{poly}(m))$, and the output BDD sizes are all exponential in m. Moreover, the output BDD sizes remain exponential in m for any order $<$ of elements since Lemma 2 holds for any order $<$ of elements. This constitutes the theoretical result that the family algebra operations in Table 1, except for the first four operations, cannot be performed in polynomial time in the input BDD sizes.

Second, we discuss how often such exponential blow-up occurs. Although we rely on specific families, the hidden weighted bit function $\mathcal{H}_m$ and the permutation function $\mathcal{P}_m$, the heart of the above proofs is that even a single operation may cause us to compute the union or intersection of multiple subfamilies. Apart from these families, it is usual that taking the union or intersection of multiple families leads to exponential blow-up. To imagine this, we consider encoding a family described by polynomial-sized conjunctive normal form (CNF) into BDD/ZDD. Every clause can be encoded into a polynomial-sized BDD/ZDD. Moreover, if the entire CNF is encoded into BDD/ZDD, we can solve SAT, or even more difficult #SAT, in linear time with respect to the size of BDD/ZDD [13]. However, it is a famous fact that SAT and #SAT are in NP-complete and #P-complete, respectively, meaning that they are believed not to be solved in polynomial time. This means that for many CNFs, the BDD/ZDD after taking intersection of clauses does not remain polynomial-sized. Therefore, apart from the specific examples used in the proof, there are many cases yielding the blow-up of BDD/ZDD size after single family algebra operation.

Finally, we argue the limitation of some of the above results that the permutation function is not such a "devilish" example. The permutation function is a family of subsets of a set with $O(m^2)$ elements and its ZDD size can only be lower bounded by $\Omega(2^m/\text{poly}(m))$. Since the ZDD size of the family of subsets of a set with $O(m^2)$ can be at most $\Omega(2^{m^2}/\text{poly}(m))$, it is far from being the worst-case. We should investigate whether there is a family of sets generated by restrict or similar operations whose ZDD size is lower bounded by $\Omega(\alpha^n/\text{poly}(n))$, where $\alpha > 1$ and n is the number of elements in the base set.

4 Conclusion

We proved that the worst-case complexity of carrying out certain kinds of a family algebra operation on BDDs/ZDDs once is lower bounded by an exponential factor. These include all of the operations raised by Knuth [13, §7.1.4 Ex. 203,204,236,243] except for the basic set operations. In particular, we resolved the controversy over the complexity of the join operation, which had arisen prominently in past literature. We also resolved the open problem regarding the worst-case complexity of the quotient operation.

Future directions include the followings. First, we only prove the lower-bound of the complexity of carrying out a single operation. It should be investigated whether we can obtain a non-trivial upper-bound of the complexity. Second, it is unknown whether a "double recursion" procedure like those in Section 2.2 always leads to an exponential worst-case complexity. It is important to investigate whether there are non-trivial operations that should require a double recursion procedure even though the worst-case complexity is polynomial.

References

1 B. Bollig. A simpler counterexample to a long-standing conjecture on the complexity of Bryant's apply algorithm. *Inf. Process. Lett.*, 114(3):124–129, 2014. doi:10.1016/j.ipl.2013.11.003.

2 R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

3 R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. Comput.*, 40(2):205–213, 1991. doi:10.1109/12.73590.

4 O. Coudert. Solving graph optimization problems with ZBDDs. In *Proc. of the European Design & Test Conference (ED&TC 97)*, pages 224–228, 1997. doi:10.1109/EDTC.1997.582363.

5 O. Coudert, J. C. Madre, and H. Fraisse. A new viewpoint on two-level logic minimization. In *Proc. of the 30th ACM/IEEE Design Automation Conference (DAC 1993)*, pages 625–630, 1993. doi:10.1145/157485.165071.

6 A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17(1):229–264, 2002. doi:10.1613/jair.989.

7 U. Gupta, P. Kalla, and V. Rao. Boolean Gröbner basis reductions on finite field datapath circuits using the unate cube set algebra. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 38(3):576–588, 2019. doi:10.1109/TCAD.2018.2818726.

8 T. Inoue, H. Iwashita, J. Kawahara, and S. Minato. Graphillion: software library for very large sets of labeled graphs. *Int. J. Softw. Tools Technol. Trans.*, 18(1):57–66, 2016. doi:10.1007/s10009-014-0352-z.

9 T. Inoue, N. Yasuda, S. Kawano, Y. Takenobu, S. Minato, and Y. Hayashi. Distribution network verification for secure restoration by enumerating all critical failures. *IEEE Trans. Smart Grid*, 6(2):843–852, 2015. doi:10.1109/TSG.2014.2359114.

10 A. Ito, R. Ueno, and N. Homma. Efficient formal verification of Galois-Field arithmetic circuits using ZDD representation of Boolean polynomials. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(3):794–798, 2022. doi:10.1109/TCAD.2021.3059924.

11 J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Trans. Fundamentals*, E100-A(9):1773–1784, 2017. doi:10.1587/transfun.E100.A.1773.

12 J. Kawahara, T. Saitoh, H. Suzuki, and R. Yoshinaka. Solving the longest oneway-ticket problem and enumerating letter graphs by augmenting the two representative approaches with ZDDs. In *Proc. of the Computational Intelligence in Information System (CIIS 2016)*, pages 294–305, 2016. doi:10.1007/978-3-319-48517-1_26.

13 D. E. Knuth. *The Art of Computer Programming: Vol. 4A. Combinatorial Algorithms, Part 1*, volume 4A: Combinatorial Algorithms, Part I. Addison-Wesley Professional, 2011.

14 S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE Design Automation Conference (DAC 1993)*, pages 272–277, 1993. doi:10.1145/157485.164890.

15 S. Minato. Calculation of unate cube set algebra using zero-suppressed BDDs. In *Proc. of the 31st ACM/IEEE Design Automation Conference (DAC 1994)*, pages 420–424, 1994. doi:10.1145/196244.196446.

16 S. Minato, T. Uno, and H. Arimura. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *Proc. of Advances in Knowledge Discovery and Data Mining (PAKDD 2008)*, pages 234–246, 2008. doi:10.1007/978-3-540-68125-0_22.

17 H. G. Okuno, S. Minato, and H. Isozaki. On the properties of combination set operations. *Inf. Process. Lett.*, 66(4):195–199, 1998. doi:10.1016/S0020-0190(98)00067-2.

18 F. Somenzi. CUDD: CU decision diagram package, 1997. URL: https://github.com/ivmai/cudd.

19 Y. Takenobu, N. Yasuda, S. Kawano, S. Minato, and Y. Hayashi. Evaluation of annual energy loss reduction based on reconfiguration scheduling. *IEEE Trans. Smart Grid*, 9(3):1986–1996, 2018. doi:10.1109/TSG.2016.2604922.

20 R. Yoshinaka, J. Kawahara, S. Denzumi, H. Arimura, and S. Minato. Counterexamples to the long-standing conjecture on the complexity of BDD binary operations. *Inf. Process. Lett.*, 112(16):636–640, 2012. doi:10.1016/j.ipl.2012.05.007.

A Fast Algorithm for Computing a Planar Support for Non-Piercing Rectangles

Ambar Pal

Johns Hopkins University, Baltime, MD, USA

Rajiv Raman

Indraprastha Institute of Information Technology Delhi, New Delhi, India

Saurabh Ray

NYU Abu Dhabi, UAE

Karamjeet Singh

Indraprastha Institute of Information Technology Delhi, New Delhi, India

Abstract

For a hypergraph $\mathcal{H} = (X, \mathcal{E})$ a *support* is a graph G on X such that for each $E \in \mathcal{E}$, the induced subgraph of G on the elements in E is connected. If G is planar, we call it a planar support. A set of axis parallel rectangles $\mathcal{R}$ forms a non-piercing family if for any $R_1, R_2 \in \mathcal{R}$, $R_1 \setminus R_2$ is connected.

Given a set P of n points in $\mathbb{R}^2$ and a set $\mathcal{R}$ of m *non-piercing* axis-aligned rectangles, we give an algorithm for computing a planar support for the hypergraph $(P, \mathcal{R})$ in $O(n \log^2 n + (n + m) \log m)$ time, where each $R \in \mathcal{R}$ defines a hyperedge consisting of all points of P contained in R.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Algorithms, Hypergraphs, Computational Geometry, Visualization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.53

1 Introduction

Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a *support* is a graph G on V such that for all $E \in \mathcal{E}$, the subgraph induced by E in G, denoted $G[E]$ is connected. The notion of a support was introduced by Voloshina and Feinberg [30] in the context of VLSI circuits. Since then, this notion has found wide applicability in several areas, such as visualizing hypergraphs [6, 7, 8, 9, 11, 19, 21], in the design of networks [1, 3, 4, 13, 20, 23, 26], and similar notions have been used in the analysis of local search algorithms for geometric problems [2, 5, 15, 24, 25, 27].

Any hypergraph clearly has a support: the complete graph on all vertices. In most applications however, we require a support with an additional structure. For example, we may want a support with the fewest number of edges, or a support that comes from a restricted family of graphs (e.g., outerplanar graphs).

Indeed, the problem of constructing a support has been studied by several research communities. For example, Du, et al., [16, 17, 18] studied the problem of minimizing the number of edges in a support, motivated by questions in the design of vacuum systems. The problem has also been studied under the topic of "minimum overlay networks" [20, 14] with applications to distributed computing. Johnson and Pollack [22] showed that it is NP-hard to decide if a hypergraph admits a planar support.

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 53; pp. 53:1–53:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In another line of work, motivated by the analysis of approximation algorithms for packing and covering problems on *geometric hypergraphs*[1], several authors have considered the problem of constructing supports that belong to a family having sublinear sized separators[2] such as planar graphs, or graphs of bounded genus [27, 28, 29]. For this class of problems, the problem of interest is only to show the existence of a support from a restricted family of graphs.

Our contribution. So far there are very few tools or techniques to construct a support for a given hypergraph or even to show that a support with desired properties (e.g., planarity) exists. Our paper presents a fast algorithm to construct a planar support for a restricted setting, namely hypergraphs defined by axis-parallel rectangles that are *non-piercing*, i.e., for each pair of intersecting rectangles, one of them contains a corner of the other. This may seem rather restrictive. However, even if we allow each rectangle to belong to at most one piercing pair of rectangles, it is not difficult to construct examples where for any $r \geq 3$, any support must have $K_{r,r}$ as a topological minor. To see this, consider a geometric drawing of $K_{r,r}$ in the usual manner, i.e., the two partite sets on two vertical lines, and the edges as straight-line segments. Replace each edge of the graph by a long path, and then replace each edge along each path by a small rectangle that contains exactly two points. Where the edges cross, a pair of rectangles corresponding to each edge cross. Since each rectangle contains two points, it leaves us no choice as to the edges we can add. It is easy to see that the resulting support contains $K_{r,r}$ as a topological minor. Further, even for this restricted problem, the analysis of our algorithm is highly non-trivial, and we hope that the tools introduced in this paper will be of wider interest.

Raman and Ray [27], showed that the hypergraph defined by *non-piercing regions*[3] in the plane admits a planar support. Their proof implies an $O(m^2(\min\{m^3, mn\} + n))$ time algorithm to compute a planar support where m is the number of regions and n is the number of points in the arrangement of the regions. While their algorithm produces a plane embedding, the edges may in general be arbitrarily complicated curves i.e., they may have an arbitrary number of bends. It can be shown that if the non-piercing regions are convex then there exists an embedding of the planar support with straight edges but it is not clear how to find such an embedding efficiently.

We present a simple and fast algorithm for drawing plane supports with straight-line edges for non-piercing rectangles. More precisely, the following is the problem definition:

Support for non-piercing rectangles:
Input: A set of m axis-parallel non-piercing rectangles $\mathcal{R}$ and a set P of n points in $\mathbb{R}^2$.
Output: A plane graph G on P s.t. for each $R \in \mathcal{R}$, $G[R \cap P]$, namely the induced subgraph on the points in $R \cap P$, is connected.

Our algorithm runs in $O(n \log^2 n + (n+m)\log m)$ time, and can be easily implemented using existing data structures. The embedding computed by our algorithm not only has straight-line edges but also for each edge e, the axis-parallel rectangle with e as the diagonal does not contain any other point of P – this makes the visualization cleaner.

[1] In a geometric hypergraph, the elements of the hypergraph are points in the plane, and the hyperedges are defined by geometric regions in the plane, where each region defines a hyperedge consisting of all points contained in the region.

[2] A family of graphs $\mathcal{G}$ admits sublinear sized separators if there exist $0 < \alpha, \beta < 1$ s.t. for any $G \in \mathcal{G}$, there exists a set $S \subseteq V(G)$ s.t. $G[V \setminus S]$ consists of two parts A and B with $|A|, |B| \leq \alpha|V|$, and there is no path in $G[V \setminus S]$ between a vertex in A to a vertex in B. Further, $|S| \leq |V|^{1-\beta}$.

[3] A family of simply connected regions $\mathcal{R}$, each of whose boundary is defined by a simple Jordan curve is called non-piercing if for every pair of regions $A, B \in \mathcal{R}$, $A \setminus B$ and $B \setminus A$ are connected. The result of [27] was for more general families.

In order to develop a faster algorithm, we need to find a new construction (different from [27]), and the proof of correctness for this construction is not so straightforward. We use a sweep line algorithm. However, at any point in time, it is not possible to have the invariant that the current graph is a support for the portions of the rectangles that lie to the left of the sweep line. Instead, we show that certain *slabs* within each rectangle induce connected components of the graph and only after we sweep over a rectangle completely do we finally have the property that the set of points in that rectangle induce a connected subgraph.

Organization. The rest of the paper is organized as follows. We start in Section 2 with related work. In Section 3, we present preliminary notions required for our algorithm. In Section 4, we present a fast algorithm to construct a planar support. We show in Section 4.1 that the algorithm is correct, i.e., it does compute a planar support. We present the implementation details in Section 4.2.

2 Related work

The notion of the existence of a support, and in particular a planar support arose in the field of VLSI design [30]. A VLSI circuit is viewed as a hypergraph where each individual electric component corresponds uniquely to a vertex of the hypergraph, and sets of components called *nets* correspond uniquely to a hyperedge. The problem is to connect the components with wires so that for every net, there is a tree spanning its components. Note that planarity in this context is natural as we don't want wires to cross.

Thus, a motivation to study supports was to define a notion of planarity suitable for hypergraphs. Unlike for graphs, there are different notions of planarity of hypergraphs, not all equivalent to each other. Zykov [32] defined a notion of planarity that was more restricted. A hypergraph is said to be Zykov-planar if its incidence bipartite graph is planar [32, 31].

Johnson and Pollack [22] showed that deciding if a hypergraph admits a planar support is NP-hard. The NP-hardness result was sharpened by Buchin, et al., [11] who showed that deciding if a hypergraph admits a support that is a k-outerplanar graph, for $k \geq 2$ is NP-hard, and showed that we can decide in polynomial time if a hypergraph admits a support that is a tree of bounded degree. Brandes, et al., [8] showed that we can decide in polynomial time if a hypergraph admits a support that is a cactus[4].

Brandes et al., [9], motivated by the drawing of metro maps, considered the problem of constructing *path-based supports*, which must satisfy an additional property that the induced subgraph on each hyperedge contains a Hamiltonian path on the vertices of the hyperedge.

Another line of work, motivated by the analysis of approximation algorithms for packing and covering problems on geometric hypergraphs started with the work of Chan and Har-Peled [12], and Mustafa and Ray [25]. The authors showed, respectively, that for the Maximum Packing[5] of pseudodisks[6], and for the Hitting Set[7] problem for pseudodisks, a simple *local search* algorithm yields a PTAS. These results were extended by Basu Roy, et

[4] A cactus is a graph where each edge of the graph lies in at most one cycle.

[5] In a Maximum Packing problem, the goal is to select the largest subset of pairwise disjoint hyperedges of a hypergraph.

[6] A set of simple Jordan curves is a set of pseudocircles if the curves pairwise intersect twice or zero times. The pseudocircles along with the bounded region defined by the curves is a collection of pseudodisks.

[7] In the Hitting Set problem, the goal is to select the smallest subset of vertices of a hypergraph so that each hyperedge contains at least one vertex in the chosen subset.

al., [5] to work for the Set Cover and Dominating Set[8] problems defined by points and non-piercing regions, and by Raman and Ray [27], who gave a general theorem on the existence of a planar support for any geometric hypergraph defined by two families of *non-piercing regions*. This result generalized and unified the previously mentioned results, and for a set of m non-piercing regions, and a set of n points in the plane, it implies that a support graph can be constructed in time $O(m^2(\min\{m^3, mn\} + n))$. It follows that for non-piercing axis-parallel rectangles, a planar support can be constructed in time $O(m^2(\min\{m^3, mn\} + n))$. However, in the embedding of the support thus constructed, the edges may be drawn as arbitrary curves.

3 Preliminaries

Let $\mathcal{R} = \{R_1, \ldots, R_m\}$ denote a set of axis-parallel rectangles and let $P = \{p_1, \ldots, p_n\}$ denote a set of points in the plane. We assume that the rectangles and points are in *general position*, i.e., the points in P have distinct x and y coordinates, and the boundaries of any two rectangles in $\mathcal{R}$ are defined by distinct x-coordinates and distinct y-coordinates. Further, we assume that no point in P lies on the boundary of a rectangle in $\mathcal{R}$.

Piercing, Discrete Piercing. A rectangle R' is said to *pierce* a rectangle R if $R \setminus R'$ consists of two connected components. A collection $\mathcal{R}$ of rectangles is *non-piercing* if no pair of rectangles pierce. A rectangle R' *discretely pierces* a rectangle R if R' pierces R and each component of $R \setminus R'$ contains a point of P. Since we are primarily concerned with discrete piercing, the phrase "R pierces R'" will henceforth mean discrete piercing, unless stated otherwise. Note that while piercing is a symmetric relation, discrete piercing is not.

"L"-shaped edge. We construct a drawing of a support graph G on P using "L"-shaped edges of type: ┐ or ┘. Henceforth, the term *edge* will mean one of the two "L"-shaped edges joining two points. The embedded graph may not be planar due to the overlap of the edges along their horizontal/vertices segments. However, as we show, G satisfies the additional property that for each edge, the axis-parallel rectangle defined by the edge has no points of P in its interior (formal definition below), and that no pair of edges cross. Consequently, replacing each edge with the straight segment joining its end-points yields a plane embedding of G.

Delaunay edge, Valid edge, $\mathbf{R(\cdot), h(\cdot), v(\cdot)}$. For an edge between points $p, q \in P$, let $R(pq)$ denote the rectangle with diagonally opposite corners p and q. The edge pq is a *Delaunay edge* if the interior of $R(pq)$ does not contain a point of P. We say that an edge pq (discretely) pierces a rectangle R if $R \setminus \{pq\}$ consists of two regions, and each region contains a point of P. An edge pq is said to be *valid* if it does not discretely pierce any rectangle $R \in \mathcal{R}$, and does not cross any existing edge. For an edge pq, we use $h(pq)$ for the horizontal segment of pq, and $v(pq)$ for the vertical segment of pq.

Monotone Path, Point above Path. A path π is said to be x-monotone if a vertical line, i.e., a line parallel to the y-axis, does not intersect the path in more than one point. We modify this definition slightly for our purposes – we say that a path consisting of a sequence

[8] In the Set Cover problem, the input is a set system $(X, \mathcal{S})$ and the goal is to select the smallest sub-collection $\mathcal{S}'$ that covers the elements in X. For a graph, a subset of vertices S is a dominating set if each vertex in the graph is either in S, or is adjacent to a vertex in S.

of $\urcorner$, or $\lrcorner$ edges is x-monotone if any vertical line intersects the path in at most one vertical segment (which may in some cases be a single point). Let π be a path and q be a point not on the path. We say that "q lies above π" if ℓ_q, the vertical line through q intersects π at point(s) below q. We define the notion that "q lies below π" analogously. Note that these notions are defined only if ℓ_q intersects π.

Left(Right)-Neighbor, Left(Right)-Adjacent. For a point $q \in P$ and a set $P' \subseteq P$, the *right-neighbor* of q in P' is q_1, where $q_1 = \text{argmin}_{q' \in P'}\{x(q') : x(q') > x(q)\}$. The *left-neighbor* of q in P' is defined similarly, i.e., q_0 is the left-neighbor of q, where $q_0 = \text{argmax}_{q' \in P'}\{x(q') : x(q') < x(q)\}$. Note that being a left- or right-neighbor is a *geometric notion*, and not related to the support graph we construct. We use the term *left-adjacent* to refer to the neighbors of q in a plane graph G that lie to the left of q. The term *right-adjacent* is defined analogously.

4 Algorithm

In this section, we present an algorithm to compute a planar support for the hypergraph defined by points and non-piercing axis-parallel rectangles in $\mathbb{R}^2$: perform a left-to-right vertical line sweep and at each input point encountered, add all possible *valid Delaunay edges* to previous points. The algorithm, presented as Algorithm 1, draws edges having shapes in $\{\urcorner, \lrcorner\}$. We prove correctness of Algorithm 1 in Section 4.1, and show how it can be implemented to run in $O(n \log^2 n + (n+m) \log m))$ time in Section 4.2.

Algorithm 1 The algorithm outputs a graph G on P embedded in $\mathbb{R}^2$, whose edges are valid Delaunay edges of type $\{\urcorner, \lrcorner\}$. Replacing each Delaunay edge $\{p, q\}$ by the diagonal of $R(pq)$ yields a plane embedding of G.

```
Input: A set $P$ of points, and a set $\mathcal{R}$ of non-piercing axis-parallel rectangles in $\mathbb{R}^2$.
Output: Embedded Planar Support $G = (P, E)$
Order $P$ in increasing order of x-coordinates: $(p_1, \ldots, p_n)$
$E = \emptyset$
for each point $p_i$ in sorted order, $i \in \{2, 3, \ldots, n\}$ do
    $E = E \cup \{e_{ij} = p_i p_j \mid j < i$, and $e_{ij}$ is a valid Delaunay edge.$\}$
end
```

4.1 Correctness

In this section, we show that the graph G constructed on P by Algorithm 1 is a support graph for the rectangles in $\mathcal{R}$, and this is sufficient as planarity follows directly by construction. The proof is technical, and we start with some necessary notation.

For a rectangle R, we denote the y-coordinates of the lower and upper horizontal sides by $y_-(R)$ and $y_+(R)$, respectively. Similarly, $x_-(R)$ and $x_+(R)$ denote respectively the x-coordinates of the left and right vertical sides. We denote the vertical line through any point p by ℓ_p.

We use $\text{Piece}(R, H)$ to denote the rectangle $R \cap H$ for a halfplane H defined by a vertical line. We abuse notation and use $\text{Piece}(R, p)$ to denote the rectangle $R \cap H_-(\ell_p)$, the intersection of R with the left half-space defined by the vertical line through the point p.

We also use the notation $R[x_-, x_+]$ to denote the sub-rectangle of rectangle R, that lies between x-coordinates x_- and x_+. Similarly, we use $R[y_-, y_+]$ to denote the sub-rectangle of R that lies between the y-coordinates y_- and y_+.

To avoid boundary conditions in the definitions that follow, we add two rectangles: R_{top} above all rectangles in $\mathcal{R}$, and R_{bot} below all rectangles in $\mathcal{R}$, that is $y_-(R_{top}) > \max_{R\in\mathcal{R}} y_+(R)$, and $y_+(R_{bot}) < \min_{R\in\mathcal{R}} y_-(R)$. The rectangles R_{top}, and R_{bot} span the width of all rectangles, i.e., $x_-(R_{top}) = x_-(R_{bot}) < \min_{R\in\mathcal{R}} x_-(R)$, and $x_+(R_{top}) = x_+(R_{bot}) > \max_{R\in\mathcal{R}} x_+(R)$. We add two points $P_{top} = \{p_1^+, p_2^+\}$ to the interior of R_{top}, and two points $P_{bot} = \{p_1^-, p_2^-\}$ to the interior of R_{bot}, such that $x(p_1^+) = x(p_1^-) < \min_{p\in P} x(p)$, and $x(p_2^+) = x(p_2^-) > \max_{p\in P} x(p)$. Let $\mathcal{R}' = \mathcal{R} \cup \{R_{top}, R_{bot}\}$, and $P' = P \cup P_{top} \cup P_{bot}$. For ease of notation, we simply use $\mathcal{R}$ and P to denote $\mathcal{R}'$ and P' respectively, and implicitly assume the existence of $R_{top}, R_{bot}, P_{top}$ and P_{bot}.

For a vertical segment s, a rectangle $R \in \mathcal{R}'$ is said to be *active* at s if it is either discretely pierced by s i.e., $R \setminus s$ is not connected and each of the two components contains a point of P, or there is a point of $P \cap s$ in R. We denote the set of all active rectangles at s by $\textsc{Active}(s)$. For a point $p \in P \cap s$, we define $\textsc{Contain}(s,p)$ to be the set of rectangles in $\textsc{Active}(s)$ that contains the point p. We define $\textsc{Above}(s,p)$ to be the set of rectangles in $\textsc{Active}(s)$ that lie strictly above p, i.e., $\textsc{Above}(s,p) = \{R \in \textsc{Active}(s) : y_-(R) > y(p)\}$. Similarly, $\textsc{Below}(s,p) = \{R \in \textsc{Active}(s) : y_+(R) < y(p)\}$. It follows that for any point $p \in s$, $\textsc{Active}(s) = \textsc{Contain}(s,p) \sqcup \textsc{Above}(s,p) \sqcup \textsc{Below}(s,p)$, where $\sqcup$ denotes disjoint union.

Note that for the vertical line ℓ_p through $p \in P$, $\textsc{Active}(\ell_p) \neq \emptyset$, as $\textsc{Active}(\ell_p)$ contains the rectangles R_{top} and R_{bot}. Similarly, $\textsc{Above}(\ell_p,p) \neq \emptyset$ and $\textsc{Below}(\ell_p,p) \neq \emptyset$. Abusing notations slightly, we write $\textsc{Active}(p)$ instead of $\textsc{Active}(\ell_p)$, and likewise with $\textsc{Contain}(\cdot), \textsc{Above}(\cdot)$ and $\textsc{Below}(\cdot)$.

For a point $p \in P$, we now introduce the notion of *barriers*. Any active rectangle R' in $\textsc{Above}(p)$ prevents a valid Delaunay edge incident on p from being incident to a point to the left of p above $y_+(R')$, as such an edge would discretely pierce R'. Hence, among all rectangles $R' \in \textsc{Above}(p)$, the one with lowest $y_+(R')$ is called the *upper barrier* at p, denoted $\textsc{Ub}(p)$. Thus, $\textsc{Ub}(p) = \operatorname{argmin}_{R'\in\textsc{Above}(p)} y_+(R')$.

Similarly, we define the *lower barrier* of p, $\textsc{Lb}(p) = \operatorname{argmax}_{R'\in\textsc{Below}(p)} y_-(R')$.

Note that $\textsc{Ub}(p)$ and $\textsc{Lb}(p)$ exist for any $p \in P$ since $\textsc{Above}(p)$ and $\textsc{Below}(p)$ are non-empty.

While the rectangles in $\mathcal{R}'$ are non-piercing, a rectangle $R' \in \textsc{Active}(p)$ can be discretely pierced by $\textsc{Piece}(R,p)$. We thus define the *upper piercing barrier* $\textsc{Upb}(R,p)$ as the rectangle $R' \in \textsc{Above}(p)$ with the lowest $y_+(R')$ that is pierced by $\textsc{Piece}(R,p)$, and we define the *lower piercing barrier* $\textsc{Lpb}(R,p)$ analogously. That is,

$$\textsc{Upb}(R,p) = \operatorname*{argmin}_{\substack{R'\in\textsc{Above}(p)\\ \textsc{Piece}(R,p)\text{ pierces }R'}} y_+(R') \quad\text{and}\quad \textsc{Lpb}(R,p) = \operatorname*{argmax}_{\substack{R'\in\textsc{Below}(p)\\ \textsc{Piece}(R,p)\text{ pierces }R'}} y_-(R')$$

For a point $p \in P$ and a rectangle $R \in \textsc{Contain}(p)$, if $\textsc{Upb}(R,p)$ or $\textsc{Lpb}(R,p)$ exist, then the horizontal line containing $y_+(\textsc{Upb}(R,p))$ together with the horizontal line containing $y_-(\textsc{Lpb}(R,p))$ naturally split $\textsc{Piece}(R,p)$ into at most three sub-rectangles called *slabs*. The point p lies in exactly one of these slabs, denoted $\textsc{Slab}(R,p)$. Thus, $\textsc{Slab}(R,p)$ is the sub-rectangle of R whose left and right-vertical sides are respectively defined by $x_-(R)$ and ℓ_p, and the upper and lower sides are respectively defined by

$$y_+(\textsc{Slab}(R,p)) = \begin{cases} y_+(\textsc{Upb}(R,p)), & \text{if } \textsc{Upb}(R,p) \text{ exists} \\ y_+(R), & \text{otherwise} \end{cases}$$

and similarly,

$$y_-(\textsc{Slab}(R,p)) = \begin{cases} y_-(\textsc{Lpb}(R,p)), & \text{if } \textsc{Lpb}(R,p) \text{ exists} \\ y_-(R), & \text{otherwise} \end{cases}$$

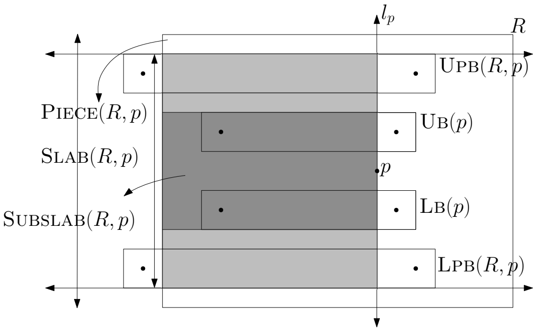


Figure 1 The figure above shows $\textsc{Ub}(p), \textsc{Lb}(p)$, and the upper and lower piercing barriers $\textsc{Lpb}(R,p)$ and $\textsc{Upb}(R,p)$ of $\textsc{Piece}(R,p)$. The slab $\textsc{Slab}(R,p)$ containing p defined by $\textsc{Upb}(R,p)$ and $\textsc{Lpb}(R,p)$ is shaded. The dark grey part shows the $\textsc{Subslab}(R,p)$.

By definition, for a point p and $R \in \textsc{Contain}(p)$, if $\textsc{Upb}(R,p)$ exists, then $y_+(\textsc{Upb}(R,p)) \geq y_+(\textsc{Ub}(p))$. Similarly, if $\textsc{Lpb}(R,p)$ exists, then $y_-(\textsc{Lpb}(R,p)) \leq y_-(\textsc{Lb}(p))$. Thus, $y_+(\textsc{Ub}(p))$ and $y_-(\textsc{Lb}(p))$ together split $\textsc{Slab}(R,p)$ further into at most 3 sub-rectangles called *sub-slabs* whose vertical sides coincide with the vertical sides of $\textsc{Slab}(R,p)$, and the horizontal sides are defined by $y_+(\textsc{Ub}(p))$ and $y_-(\textsc{Lb}(p))$. Let $\textsc{Subslab}(R,p)$ denote the sub-slab containing p. Figure 1 illustrates the notions defined thus far. Note that the left-adjacent vertices of p in G that are contained in R, only lie in $\textsc{Subslab}(R,p)$.

Proof Strategy. To prove that the graph G constructed by Algorithm 1 is a support for $\mathcal{R}$, we proceed in two steps. First (and the part that requires most of the work) we show that for each $R \in \mathcal{R}$ and $p \in P \cap R$, the subgraph of G induced by the points in $\textsc{Slab}(R,p)$ is connected. Second, we show that if p is the rightmost point in R, then $\textsc{Slab}(R,p)$ contains all points in $R \cap P$ which, by the first part, is connected.

When processing a point p, Algorithm 1 only adds valid Delaunay edges from p to points to its left. That is, we only add edges to a subset of points in $\textsc{Subslab}(R,p)$. To show that $\textsc{Slab}(R,p)$ is connected, one approach could be to show that the $\textsc{Slab}(R,p)$ is covered by sub-slabs defined by points in $\textsc{Slab}(R,p)$, adjacent sub-slabs share a point of P, and that points in a sub-slab induce a connected subgraph. Unfortunately, this is not true, and we require a finer partition of a slab. We proceed as follows: First, we define a sequence of sub-rectangles of $\textsc{Slab}(R,p)$ called *strips*, denoted $\textsc{Strip}(R,p,i)$ for $i \in \{-t,\ldots,k\}$, where the strips that lie above p have positive indices, the strips that lie below p have negative indices, and the unique strip that contains p has index 0. Further, each strip shares its vertical sides with $\textsc{Slab}(R,p)$. In the following, since R and p are fixed, we refer to $\textsc{Strip}(R,p,i)$ as $\textsc{Strip}_i$. We define the strips so that they satisfy the following conditions:

(s-i) Each strip is contained in the slab, i.e, $\textsc{Strip}_i \subseteq \textsc{Slab}(R,p)$ for each $i \in \{-t,\ldots,k\}$.

(s-ii) The union of strips cover the slab, i.e., $\textsc{Slab}(R,p) \subseteq \cup_{i=-t}^{k} \textsc{Strip}_i$, and

(s-iii) Consecutive strips contain a point of P in their intersection, i.e., $\textsc{Strip}_i \cap \textsc{Strip}_{i-1} \cap P \neq \emptyset$ for all $i \in \{-t+1,\ldots,k\}$. Consequently, each strip contains a point of P.

In order to prove that $\textsc{Slab}(R,p)$ is connected, we describe below a strategy that does not quite work but, as we show later, can be fixed.

Let $\textsc{Strip}_i \cap P = P_i$. By Condition (s-iii), $P_i \neq \emptyset$ for any $i \in \{-t,\ldots,k\}$. For a strip $\textsc{Strip}_i$, let p_i denote the rightmost point in it. Let us assume for now that for each $i \in \{-t,\ldots,k\}$, and each point $q \in P_i$, there is a path from q to p_i that lies entirely in $\textsc{Strip}_i$.[9] Now, consider an arbitrary point $q \in \textsc{Slab}(R, p)$. By Condition (s-ii) each point in $P \cap \textsc{Slab}(R, p)$ is contained in at least one strip. Therefore, $q \in \textsc{Strip}_i$ for some $i \in \{-t, \ldots, k\}$. By our assumption, there is a path π_i^1 from q to p_i that lies entirely in $\textsc{Strip}_i$. If $i \geq 0$ (a symmetric argument works when $i < 0$), since Condition (s-iii) implies consecutive strips intersect at a point in P, there is a path π_i^2 from p_i to a point $q' \in P_i \cap P_{i-1}$ that lies entirely in $\textsc{Strip}_i$. Again, by our assumption, there is a path π_{i-1}^1 from q' to p_{i-1} that lies entirely in $\textsc{Strip}_{i-1}$. Repeating the argument above with $i-1, i-2, \ldots$, until $i = 0$, and concatenating the paths $\pi_i^1, \pi_i^2, \pi_{i-1}^1, \ldots$, we obtain a path π from q to p, each sub-path of which is a path from a point in a strip to the rightmost point in that strip such that each point in the path lies entirely in the strip. By Condition (s-i), $\textsc{Strip}_i \subseteq \textsc{Slab}(R, p)$ for each $i \in \{-t, \ldots, k\}$. Therefore, π lies entirely in $\textsc{Slab}(R, p)$. Since q was arbitrary, this implies that $\textsc{Slab}(R, p)$ is connected.

Consider a slab $\textsc{Slab}(R, p)$ corresponding to a rectangle $R \in \mathcal{R}$ and a point $p \in P \cap R$. The strips corresponding to $\textsc{Slab}(R, p)$ are defined as follows: Let s denote the open segment of ℓ_p between p and $y_+(\textsc{Slab}(R, p))$ of ℓ_p, the vertical line through p. Let $\mathcal{R}_s = (R_0, \ldots, R_h)$ be the rectangles in $\textsc{Active}(s)$ ordered by their upper sides i.e., $y_+(R_i) < y_+(R_j)$, for $0 \leq i < j \leq h$. Similarly, let s' denote the open segment of ℓ_p between p and $y_-(\textsc{Slab}(R, p))$ and let $\mathcal{R}_{s'} = (R'_0, \ldots, R'_{h'})$ denote the rectangles in $\textsc{Active}(s')$ ordered by their lower sides $y_-(R'_i) > y_-(R'_j)$ for $0 \leq i < j \leq h'$.

We define $\textsc{Strip}_0 = \textsc{Slab}(R, p)[y_-(R'_0), y_+(R_0)]$, if $\textsc{Active}(s) \neq \emptyset$ and $\textsc{Active}(s') \neq \emptyset$. If $\textsc{Active}(s) = \emptyset$, we set $y_+(\textsc{Strip}_0) = y_+(\textsc{Slab}(R, p))$. Similarly, if $\textsc{Active}(s') = \emptyset$, we set $y_-(\textsc{Strip}_0) = y_-(\textsc{Slab}(R, p))$. We set $p_0 = p$. Having defined $\textsc{Strip}_0$, we set $\mathcal{R}_s = \mathcal{R}_s \setminus R_0$ and $\mathcal{R}_{s'} = \mathcal{R}_{s'} \setminus R'_0$.

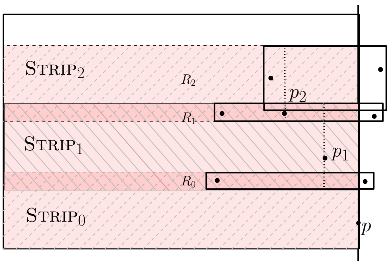


Figure 2 The figure shows the construction of the strips $\textsc{Strip}_0$, $\textsc{Strip}_1$ and $\textsc{Strip}_2$. The vertical line segment through p_i, $i \in \{1, 2\}$ shows that p_i is the rightmost point among the points in the strip i.

For $i > 0$, having constructed $\textsc{Strip}_j$ for $j = 0, \ldots, i-1$, we do the following while $\mathcal{R}_s \neq \emptyset$: Let $S_i = \operatorname{argmin}_{R' \in \mathcal{R}_s} y_-(R')$, and let $y_- = y_-(S_i)$. Let $R_i = \operatorname{argmin}\{y_+(R') : R' \in \mathcal{R}_s : y_-(R') > y_-\}$, and let $y_+ = \min\{y_+(\textsc{Slab}(R, p)), y_+(R_i)\}$. Set $y_-(\textsc{Strip}_i) = y_-$ and $y_+(\textsc{Strip}_i) = y_+$. Let $p_i = \operatorname{argmax}\{x(p') : p' \in P \cap \textsc{Slab}(R, p) : y_- < y(p') < y_+\}$. Note that p_i exists since $S_i \in \textsc{Active}(s)$. Set $\mathcal{R}_s = \mathcal{R}_s \setminus \{R' : y_-(R') < y_-(R_i)\}$.

For $i < 0$, the construction is symmetric. Having constructed $\textsc{Strip}(R, p, j)$ for $j = 0, -1, \ldots, -i+1$, we do the following until $\mathcal{R}_{s'} = \emptyset$. Let $S'_i = \operatorname{argmax}_{R' \in \mathcal{R}_{s'}} y_+(R')$, and let $y_+ = y_+(S'_i)$. Let $R'_i = \operatorname{argmax}\{y_-(R') : R' \in \mathcal{R}_{s'}, y_+(R') < y_+\}$. Let $y_- = \max\{y_-(R'_i), y_-(\textsc{Slab}(R, p))\}$. Set $y_-(\textsc{Strip}_i) = y_-$ and $y_+(\textsc{Strip}_i) = y_+$. Let $p_i = \operatorname{argmax}\{x(p') : p' \in P \cap \textsc{Slab}(R, p), y_- < y(p') < y_+\}$. Again, p_i exists since $S'_i \in \textsc{Active}(s')$. Set $\mathcal{R}_{s'} = \mathcal{R}_{s'} \setminus \{R' \in \mathcal{R}_{s'} : y_+(R') > y_+(R'_i)\}$. Figure 2 illustrates the construction of the strips.

[9] This assumption is incorrect but will be remedied later.

▶ **Proposition 1.** *For $i \in \{-t, \ldots, k\}$,*

$$y_-(\textsc{Lb}(p_i)) \leq y_-(\textsc{Strip}_i) < y_+(\textsc{Strip}_i) \leq y_+(\textsc{Ub}(p_i))$$

Proof. Fix $i \in \{-t, \ldots, k\}$ and assume $i \geq 0$. For $i < 0$, the proof is symmetric. Since $p_i \in \textsc{Strip}_i$ and by the definition of the lower barrier, $y_+(\textsc{Lb}(p_i)) < y(p_i) < y_+(\textsc{Strip}_i)$. If $y_-(\textsc{Lb}(p_i)) > y_-(\textsc{Strip}_i)$, since $\textsc{Lb}(p_i) \in \mathcal{R}_s$ and $y_+(\textsc{Strip}_i) \leq \min\{y_+(R') \in \mathcal{R}_s : R' \in \mathcal{R}_s \text{ and } y_-(R') > y_-(\textsc{Strip}_i)\}$, it implies $y_+(\textsc{Strip}_i) \leq y_+(\textsc{Lb}(p_i))$, contradicting $p_i \in \textsc{Strip}_i$.

Now we argue about $\textsc{Ub}(p_i)$. If $y_+(\textsc{Ub}(p_i)) > y_+(\textsc{Slab}(R,p))$, since $y_+(\textsc{Slab}(R,p)) \geq y_+(\textsc{Strip}_i)$, we have $y_+(\textsc{Ub}(p_i)) \geq y_+(\textsc{Strip}_i)$. Otherwise, we have $\textsc{Ub}(p_i) \in \mathcal{R}_s$. Since $p_i \in \textsc{Strip}_i$ and by definition of the upper barrier, we have $y_-(\textsc{Ub}(p_i)) > y(p_i) > y_-(\textsc{Strip}_i)$. Since $y_+(\textsc{Strip}_i) \leq \min\{y_+(R') : R' \in \mathcal{R}_s \text{ and } y_-(R') > y_-(\textsc{Strip}_i)\}$, it follows that $y_+(\textsc{Strip}_i) \leq y_+(\textsc{Ub}(p_i))$. ◀

▶ **Lemma 2.** *The strips constructed as above satisfy the following conditions: (i) $\textsc{Strip}_i \subseteq \textsc{Slab}(R,p)$ for each $i \in \{-t, \ldots, k\}$. (ii) $\textsc{Slab}(R,p) = \cup_{i=-t}^{k} \textsc{Strip}_i$, and (iii) $\textsc{Strip}_i \cap \textsc{Strip}_{i-1} \cap P \neq \emptyset$ for all $i \in \{-t+1, \ldots, k\}$.*

Proof. Item (i) and (ii) follow directly by construction. For (iii), note that adjacent strips contain a piece of an active rectangle and hence their intersection contains a point of P. ◀

Unfortunately, our assumption that every point in a strip has a path to its rightmost point in the strip is not correct. To see this, consider a strip that is pierced by a rectangle R', whose intersection with the strip does not contain a point of P. Therefore, a point in the strip that lies to the left of R' cannot have a path to p_i that lies in the strip unless some of its edge is allowed to pierce R'. In order to remedy this situation, we introduce the notion of a *corridor*. A corridor corresponding to a strip is a region of $\textsc{Slab}(R,p)$ that contains all points in the strip, and such that each point in the strip has a path to the rightmost point in it that lies entirely in the corridor. Since each corridor lies in $\textsc{Slab}(R,p)$, the proof strategy can be suitably modified to show that $\textsc{Slab}(R,p)$ is connected.

We now define the corridors associated with each strip. Recall that $G = (P, E)$ is the graph constructed by Algorithm 1. For a point $q \in P$, recall that the neighbors of q in G that lie to its left are called its *left-adjacent* points. If q lies on a path π, and q' is the left-adjacent point of q on π, then we say that q' is left-adjacent to q on π. We start with the following proposition that will be useful in constructing the corridors.

▶ **Proposition 3.** *For a point $q \in P$, if $(q_1, \ldots, q_r, q_{r+1}, \ldots, q_s)$ is the sequence of its left-adjacent points in G s.t. $y(q_1) > \ldots > y(q_r) > y(q)$ and $y(q_{r+1}) < \ldots < y(q_s) < y(q)$. Then, for $1 \leq i < j \leq r$, $x(q_i) > x(q_j)$, and for $r+1 \leq i < j \leq s$, $x(q_i) < x(q_j)$.*

Proof. This follows directly from the fact that each edge in G is Delaunay. ◀

For a strip $\textsc{Strip}_i$, we define its corresponding corridor $\textsc{Corridor}_i$ as follows: The corridor is the region of $\textsc{Slab}(R,p)$ bounded by two paths: an *upper path* π_i^u, and a *lower path* π_i^ℓ, defined as follows.

The upper path $\pi_i^u = (q_0, q_1, q_2, \ldots,)$ is constructed by starting with $j = 0, q_0 = p_i$, and repeating (1) set $q_{j+1} \leftarrow q'$ where q' has the highest $y(q')$ among the left-adjacent points of q_j. (2) $j \leftarrow j+1$. We stop when we cannot find such a q' for the current q_j in G that lies in $\textsc{Slab}(R,p)$, where we complete the path by following the edge to the left-adjacent vertex q_{j+1} of q_j with highest y-coordinate. Thus, the last vertex of π_i^u possibly does not lie in $\textsc{Slab}(R,p)$.

The lower path $\pi_i^\ell = (q'_0, \ldots,)$ is constructed similarly. For $j = 0$, set $q'_0 = p_i$, and repeating (1) set $q'_{j+1} \leftarrow q'$, where q' has the *lowest* $y(q')$ among the left-adjacent points of q'_j. We stop when we cannot find such a q' in $\textsc{Slab}(R, p)$ for the current q'_j, where we complete the path by following the edge from q'_j its left-adjacent vertex q'_{j+1} with smallest y-coordinate. Thus, the last vertex of π_i^ℓ possibly does not lie in $\textsc{Slab}(R, p)$.

$\textsc{Corridor}_i$ is the region of $\textsc{Slab}(R, p)$ that lies between the upper and lower paths π_i^u and π_i^ℓ. Figure 3 shows a corridor corresponding to a strip. We start with some basic observations about the corridors thus constructed.

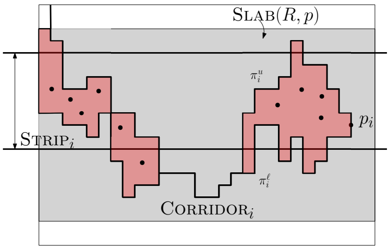


Figure 3 The figure above shows the strip $\textsc{Strip}_i$, the slab $\textsc{Slab}(R, p)$ in grey, and the corridor $\textsc{Corridor}_i$ as the region shaded in red between π_i^u and π_i^ℓ.

▶ **Proposition 4.** *For $i \in \{-t, \ldots, k\}$, π_i^u and π_i^ℓ are x-monotone.*

Proof. This follows directly by construction since at each step we augment the path by adding to it, the left-adjacent neighbor to the current vertex of the path. ◀

The graph G constructed by Algorithm 1 do not cross. We say that two paths π_1 and π_2 in G *cross* if there is an x-coordinate at which π_1 lies above π_2, and an x-coordinate at which π_2 lies above π_1.

▶ **Proposition 5.** *For $i \in \{-t, \ldots, k\}$, π_i^l does not lie above π_i^u, and π_i^ℓ and π_i^u do not cross.*

Proof. Let $\pi_i^u = (q_0, q_1, \ldots, q_r)$ and $\pi_i^\ell = (q'_0, \ldots, q'_s)$, where $q_0 = q'_0 = p_i$. Since q_1 is the left-adjacent of p_i in $\textsc{Slab}(R, p)$ with highest y-coordinate and q'_1 is the left-adjacent point of p_i with lowest y-coordinate, $y(q'_1) \leq y(q_1)$. Thus, π_i^ℓ does not lie above π_i^u at $x(p_i)$. If at some x-coordinate x', π_i^ℓ lies above π_i^u, then the paths must have crossed to the right of x'.

Let $q_i = q'_j = q$ be a point of P common to π_i^u and π_i^ℓ lying to the left of p_i. Again, since $q_{i+1} = \arg\max\{y(q'') : q'' \in \textsc{Slab}(R, p) \cap P, x(q'') < x(q), \{q, q''\} \in E(G)\}$, and $q'_{j+1} = \arg\min\{y(q'') : q'' \in \textsc{Slab}(R, p) \cap P, x(q'') < x(q), \{q, q''\} \in E(G)\}$, it follows that $y(q'_{j+1}) \leq y(q_{i+1})$. Hence, the paths do not cross, and since π_i^ℓ does not lie above π_i^u at $x(p_i)$, it does not do so at any x-coordinate to the left of p_i either. ◀

Recall that the left-neighbor of a point q in a set P' is the point $p' = \text{argmax}_{p'' \in P'}\, x(p'') < x(q)$. The right-neighbor is defined similarly. Note that the left and right neighbors are defined geometrically, and they may not be adjacent to q in the graph G. For a point $q \in P$ and $i \in \{-t, \ldots, k\}$, we let r_0 and r_1 denote respectively, the left- and right-neighbors of q on π_i^u. Similarly, we let r'_0 and r'_1 denote respectively, the left- and right-neighbors of q on π_i^ℓ.

▶ **Proposition 6.** *For $i \in \{-t, \ldots, k\}$ and $q \in P$, if q lies above π_i^u, then $y(q) > \max\{y(r_0), y(r_1)\}$. Similarly, if q lies below π_i^ℓ, then $y(q) < \min\{y(r'_0), y(r'_1)\}$.*

Proof. We assume q lies above π_i^u. The other case follows by an analogous argument. By Proposition 4, since π_i^u is x-monotone, it follows that r_0 and r_1 are consecutive along π_i^u, and thus, r_0r_1 is a valid Delaunay edge in G. If either $y(r_0) > y(q) > y(r_1)$, or $y(r_0) < y(q) < y(r_1)$, then it contradicts the fact that r_0r_1 is Delaunay. Hence, $y(q) > \max\{y(r_0), y(r_1)\}$ as q lies above π_i^u. ◀

We now show that the corridors constructed satisfy the required conditions. The first condition below, follows directly from construction.

▶ **Lemma 7.** *For $i \in \{-t, \dots, k\}$, $\textsc{Corridor}_i \subseteq \textsc{Slab}(R, p)$.*

Proof. Follows directly by construction. ◀

Next, we show that for each strip, its corresponding corridor contains all its points, that is all points in $P \cap \textsc{Strip}_i$ are contained between the upper and lower paths of $\textsc{Corridor}_i$. Before we do that, we need the following two technical statements.

▶ **Proposition 8.** *Let $q, q' \in P$, with $x(q) < x(q')$ s.t. qq' is Delaunay. If $qq' \notin E(G)$, then either (i) $h(qq')$ pierces a rectangle, or crosses an existing edge, or (ii) $v(qq')$ pierces a rectangle. In particular, $v(qq')$ does not cross an existing edge.*

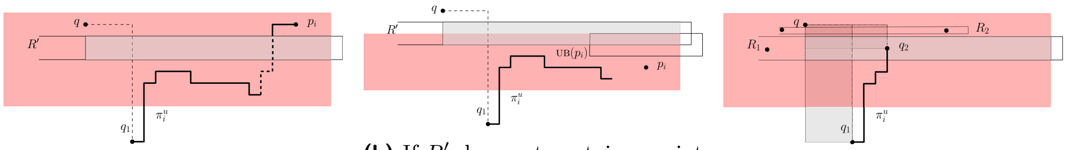
Proof. The points in P are processed by Algorithm 1 in increasing order of their x-coordinates, and when a point is being processed, we add edges of type {┐, ┘} to points to its left. Therefore, while processing q', no edge from points of P to the right of q' have been added. Hence, $v(qq')$ does cross an existing edge. ◀

▶ **Lemma 9.** *For $i \in \{-t, \dots, k\}$, let $q \in \textsc{Slab}(R, p) \cap P$ s.t. q lies above π_i^u. Let q_1 be the right-neighbor of q on π_i^u. If qq_1 is Delaunay but not valid, then $v(qq_1)$ pierces a rectangle. In particular, $h(qq_1)$ does not pierce a rectangle or cross an edge. Similarly, let $q' \in \textsc{Slab}(R, p) \cap P$ s.t. q' lies below π_i^ℓ, q_1' is the right-neighbor of q' on π_i^ℓ. If $q'q_1'$ is Delaunay but not valid, then $v(q'q_1')$ pierces a rectangle. In particular $h(q'q_1')$ does not pierce a rectangle or cross an edge.*

Proof. We prove the case when q lies above π_i^u. The other case follows by an analogous argument. Since qq_1 is not valid, either the horizontal segment of qq_1 pierces a rectangle, or crosses an existing edge; or $v(qq_1)$ pierces a rectangle since by Proposition 8, $v(qq_1)$ does not cross an existing edge.

Let q_0 be left-adjacent to q_1 on π_i^u. By Proposition 6, $y(q) > \max\{y(q_0), y(q_1)\}$. Hence qq_1 is of type ┐. Suppose $h(qq_1)$ pierces a rectangle or crosses an edge of type ┐. Then, there is a point z that lies below the $h(qq_1)$. But, z cannot lie below the $h(q_0q_1)$, as that contradicts the fact that q_0q_1 is valid. Hence, z lies above $h(q_0q_1)$. This implies that z lies either in $R(q_0q_1)$ (if $y(q_0) < y(q_1)$), or z lies in $R(qq_1)$. If $z \in R(q_0q_1)$, it contradicts the fact that q_0q_1 is Delaunay. Also, $z \notin R(qq_1)$, as qq_1 is Delaunay by assumption. Therefore, $h(qq_1)$ does not pierce a rectangle, or crosses an edge of type ┐. If $h(qq_1)$ crossed an edge e of type ┘, then either qq_1 is not Delaunay, violating our assumption, or e is not Delaunay, a contradiction. ◀

▶ **Lemma 10.** *For $i \in \{-t, \dots, k\}$, $P_i \subseteq \textsc{Corridor}_i$, where $P_i = P \cap \textsc{Strip}_i$.*



(a) R' cannot be empty if p_i is above R'.

(b) If R' does not contain a point of P between $x(q)$ and $x_+(R)$, then $y(q) > y_+(\textsc{Strip}_i)$.

(c) Since qq_1 and qq_2 are Delaunay, then R_2 pierces $v(qq_1)$.

Figure 4 The three cases in the proof showing that $P_i \subseteq \textsc{Corridor}_i$.

Proof. Suppose $P_i \setminus \textsc{Corridor}_i \neq \emptyset$. By Proposition 5, any such point either lies above π_i^u or below π_i^ℓ. We assume the former. The latter follows by an analogous argument. Let $P_i' = \{q \in \textsc{Slab}(R,p) : y(q) < y_+(\textsc{Strip}_i)$ and q lies above $\pi_i^u\}$. It suffices to show that $P_i' = \emptyset$. For the sake of contradiction, suppose $P_i' \neq \emptyset$.

We impose the following partial order on P_i: for $a, a' \in P_i$, $a \prec a' \Leftrightarrow x(a) > x(a') \wedge y(a) < y(a')$. Let q be a minimal element in P_i' according to $\prec$. In the following, when we refer to a minimal element, we impicitly assume this partial order.

We show that there is a valid Delaunay edge between q and a point q' on π_i^u. By assumption, q lies above π_i^u. Let q_0 and q_1 denote, respectively, the left- and right-neighbors of q on π_i^u.

Since q is minimal in P_i', qq_1 is Delaunay. By Proposition 6, it follows that $y(q) > y(q_0)$ and $y(q) > y(q_1)$. Since q is not left-adjacent to q_1 on π_i^u, it implies qq_1 is not valid.

Since qq_1 is Delaunay but not valid, by Lemma 9, $v(qq_1)$ pierces a rectangle. Let $\mathcal{R}'$ denote the set of rectangles pierced by qq_1. Suppose $\exists R' \in \mathcal{R}'$ s.t. $R'[x(q_1), \min\{x(p), x_+(R')\}] \cap P = \emptyset$. Then, we call R' a *bad* rectangle. Otherwise, we say that R' is *good*. Now we split the proof into two cases depending on whether $\mathcal{R}'$ contains a bad rectangle or not. In the two cases below, we use Proposition 4 that π_i^u is x-monotone.

Case 1. $\mathcal{R}'$ contains a bad rectangle. Let $R' \in \mathcal{R}'$ be a bad rectangle. First, observe that $x_+(R') > x(p)$ as otherwise, R' is not pierced by $v(qq_1)$. Now, suppose $y(p_i) > y_+(R')$, where p_i is the rightmost point in $\textsc{Strip}_i$. We have that $x(q_1) < x(p_i)$, both p_i and q_1 lie on π_i^u and, π_i^u is x-monotone. But, this implies π_i^u pierces R'. But this is impossible as by construction π_i^u consists of valid Delaunay edges. Hence, since R' is bad, we can assume that $y(p_i) < y_-(R')$. But the definition of the upper barrier implies that $y_+(R') \geq y_+(\textsc{Ub}(p_i))$. Since $v(qq_1)$ pierces R', it implies $y(q) > y_+(R')$, and hence $y(q) > y_+(\textsc{Ub}(p_i))$. But, this contradicts the assumption that $q \in \textsc{Strip}_i$, since by Proposition 1, $y_+(\textsc{Ub}(p_i)) \geq y_+(\textsc{Strip}_i)$ and hence $y(q) > y_+(\textsc{Strip}_i)$.

Case 2. All rectangles in $\mathcal{R}'$ are good. Let $R_1 = \arg\max\{y_-(R') : R' \in \mathcal{R}'\}$. Let q_2 be the leftmost point in R_1 s.t. $x(q_2) > x(q_1)$. Since q_2 lies to the right, and below q, $q_2 \prec q$. Since q is a minimal element in P_i', it implies that q_2 lies on or below π_i^u. We claim that q_2 cannot lie below π_i^u. Suppose it did. Let q_2' be the left-neighbor of q_2 on π_i^u. Then, $x(q_2') < x(q_2)$. Since q_2 is the leftmost point of R_1 to the right of $v(qq_1)$, then either $y(q_2') < y_-(R_1)$, or $y(q_2') > y_+(R_1)$. However, in either case, we obtain that π_i^u must cross R_1 between q_2 and q_1, which implies that π_i^u pierces R_1, as π_i^u is x-monotone, and the edges in π_i^u are of the form $\{\urcorner, \lrcorner\}$, a contradiction.

Since q is minimal, qq_2 is Delaunay. By Lemma 9, the only reason qq_2 is not valid is that $v(qq_2)$ pierces a rectangle. But, any such rectangle R_2 is also pierced by $v(qq_1)$, as qq_2 is Delaunay. But, this implies $y_-(R_2) > y_-(R_1)$, contradicting the choice of R_1. Therefore, qq_2 is a valid Delaunay edge. Now, the only reason that q is not the left-adjacent point of q_2

on π_i^u then, is that q_2', the left-adjacent point of q_2 on π_i^u lies in $\textsc{Slab}(R, p)$, but above q, i.e., $y(q_2') > y(q)$, as the construction of π_i^u dictates that we choose the left-adjacent point with highest y-coordinate that lies in $\textsc{Slab}(R, p)$. Showing that this leads to a contradiction completes the proof.

So suppose $y(q_2') > y(q)$, then $y(q_2') > y_+(R_1)$. Further, by Proposition 3, $x(q_2') > x(q)$. Again this implies the x-monotone curve π_i^u cannot contain both q_2' and q_1 without piercing R_1. Hence, $y(q_2') < y_-(R_1) < y(q)$, but this contradicts the choice of q_2' as the left-adjacent point of q_2 on π_i^u since qq_2 is a valid Delaunay edge with $y(q_2') < y(q_2) < y(q)$. See Figure 4 for the different cases in this proof. ◀

The key property of a corridor is that if the upper or lower path of a corridor crosses a rectangle R', then there must be a point of $R' \cap P$ that lies on that path of the corridor. Using this, we can show that any point in the strip has an adjacent point to its right in G that lies in the corridor. This implies that every point in a strip has a path to the rightmost point in the strip that lies entirely in its corresponding corridor.

▶ **Lemma 11.** *For each $q \in \textsc{Corridor}_i$, there is a path $\pi(q, p_i)$ between q and p_i that lies in $\textsc{Corridor}_i$, where $p_i \in P_i$ is the rightmost point in $\textsc{Strip}_i$.*

Proof. If q lies on the upper path π_i^u or the lower path π_i^ℓ defining $\textsc{Corridor}_i$, the lemma is immediate. So we can assume by Proposition 5 that q lies below π_i^u, and above π_i^ℓ. Suppose the lemma is false. Let q be the rightmost point of $\textsc{Corridor}_i$ that does not have a path to p_i lying in $\textsc{Corridor}_i$. To arrive at a contradiction, it is enough to show that q is adjacent to a point $q' \in \textsc{Corridor}_i$ that lies to the right of q.

Starting from ℓ_q, the vertical line through q, sweep to the right until the first point r that lies on both π_i^u and π_i^ℓ. Such a point exists since p_i lies on both π_i^u and π_i^ℓ. Let Q_i' denote the set of points in $\textsc{Corridor}_i$ whose x-coordinates lie between $x(q)$ and $x(r)$. This set is non-empty as it contains q and r. Hence, either $Q_i^+ = \{q' \in Q_i' : y(q') > y(q)\} \neq \emptyset$, or $Q_i^- = \{q' \in Q_i' : y(q') < y(q)\} \neq \emptyset$. If both are non-empty, let Q_i denote the set that contains a point with smallest x-coordinate. Otherwise, we let Q_i denote the unique non-empty set. Assume $Q_i = Q_i^+$. An analgous argument holds when $Q_i = Q_i^-$.

Define a partial order on Q_i, where for $a, b \in Q_i$, $a \prec b \Leftrightarrow x(a) < x(b)$ and $y(a) < y(b)$. Let $Q_i^{\min} = (q_1, \ldots, q_t)$ denote the sequence of minimal elements of Q_i ordered linearly such that $y(q_k) > y(q_j)$ for $k < j$. It follows that $x(q_k) < x(q_j)$ for $k < j$. Observe that qq_i is Delaunay for $i = 1, \ldots, t$ by the minimality of q_i. Our goal is to show that qq_i is a valid Delaunay edge for some $i \in \{1, \ldots, t\}$. We start with the following claim that $v(qq_t)$ and $h(qq_1)$ do not pierce a rectangle in $\mathcal{R}$, or cross an edge of G.

▷ Claim 12. $h(qq_1)$ does not pierce a rectangle in $\mathcal{R}$ or cross an edge of G, and $v(qq_t)$ does not pierce a rectangle in $\mathcal{R}$ or cross an edge of G.

Proof. Suppose $h(qq_1)$ pierced a rectangle R'. Since π_i^u consists of valid Delaunay edges, and the choice of Q_i, R' contains a point a that lies in $\textsc{Corridor}_i$. Since $h(qq_1)$ pierces R', $x(q) < x(a) < x(q_1)$. If $y(a) < y(q_1)$, then it contradicts the fact that q_1 is minimal, and if $y(a) > y(q_1)$, it contradicts the fact that q_1 is the minimal element with highest y-coordinate. A similar argument shows that $h(qq_1)$ does not cross an edge of G.

If $v(qq_t)$ pierced a rectangle $R' \in \mathcal{R}$, then R' has a point to the right of $v(qq_t)$. Further, by Proposition 4, π_i^u and π_i^ℓ are x-monotone paths and by construction, they consist of valid Delaunay edges meeting at r. If $r = q_t$ and $v(qq_t)$ pierced a rectangle, since qq_t is Delaunay, and the edges are of type $\{\urcorner, \lrcorner\}$, it implies that the left-adjacent point of r on π_i^u or π_i^ℓ is not a valid Delaunay edge. Hence, we can assume $q_t \neq r$. Again, since the edges of π_i^u and

π_i^ℓ are valid Delaunay edges, it implies that R' has a point a s.t. $x(q_t) < x(a) \le x(r)$, and a lies in $\textsc{Corridor}_i$. Since $v(qq_t)$ pierces R', $y(a) < y(q_t)$. But this contradicts the fact that q_t is the point in $Q_i^{\min}$ with the smallest y-coordinate. Therefore, $v(qq_t)$ can not pierce any rectangle. Since qq_t is Delaunay, by Proposition 8, $v(qq_t)$ does not cross an edge of G. ◁

We now define a point x_1 on the x-axis and a point y_1 on the y-axis as follows:

$$x_1 = \min\{\{x_+(R') : R' \text{ pierced by } h(qq_t)\}, \{v(e) : e \text{ crosses } h(qq_t)\}\}$$
$$y_1 = \min\{\{y_+(R') : R' \text{ pierced by } v(qq_1)\}, \{h(e) : e \text{ crosses } v(qq_1)\}\}$$

By Claim 12 and the assumption that qq_t and qq_1 are not valid edges, it follows that x_1 and y_1 exist. We argue when x_1 and y_1 correspond to $x_+(R')$ and $y_+(R'')$, respectively for rectangles $R', R'' \in \mathcal{R}$. If they were instead defined by the vertical/horizontal side of edges, the arguments are similar.

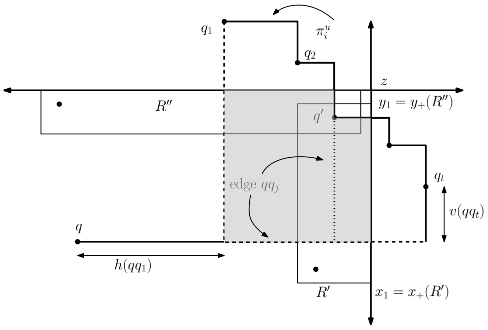


Figure 5 The edge qq' is a valid Delaunay edge.

Observe that $x_-(R'') < x(q)$, while $x_-(R') > x(q)$, and $y_-(R'') > y(q)$ and $y_-(R') < y(q)$. Now, from the fact that the rectangles are non-piercing, it implies that either $x_+(R'') < x_1$ or $y_+(R') < y_1$. Suppose wlog, the former is true. Since R' is pierced by $h(qq_t)$ and π_i^u consists of valid Delaunay edges, there are points in R' that lie in $\textsc{Corridor}_i$, and these points lie below y_1.

Let z denote the intersection of the vertical line through x_1 and the hoizontal line through y_1. By the argument above, the rectangle with diagonal qz contains points of P, and hence a point $q' \in Q_i^{\min}$. We claim that qq' is a valid Delaunay edge. To see this, note that $h(qq')$ does not pierce a rectangle in $\mathcal{R}$ as such a rectangle contradicts the definition of x_1. If $v(qq')$ pierced a rectangle, such a rectangle $\tilde{R}$ must have $y_+(\tilde{R}) < y_1$, as qq' is Delaunay. This contradicts the choice of y_1. Therefore, qq' is a valid Delaunay edge. ◀

The lemma below follows the description in the proof strategy at the start of this section.

▶ **Lemma 13.** *For a rectangle R and point $p \in R$, after Algorithm 1 has processed point p, the points in $\textsc{Slab}(R,p)$ induce a connected subgraph, all of whose edges lie in $\textsc{Slab}(R,p)$.*

Proof. Let $G[\textsc{Slab}(R,p)]$ denote the induced subgraph of G on the points in $\textsc{Slab}(R,p)$. By Condition (ii) of Lemma 2, since $\textsc{Slab}(R,p) \subseteq \cup_{i=-t}^{k}\textsc{Strip}_i$, each point in $P \cap \textsc{Slab}(R,p)$ is contained in $\cup_{i=-t}^{k}\textsc{Strip}_i$. If the statement of the lemma does not hold, consider an extremal strip, i.e., the smallest positive index, or largest negative index of a strip such that it contains a point q that does not lie in the connected component of $G[\textsc{Slab}(R,p)]$ containing p. Assume without loss of generality that $i \ge 0$. An analogous argument holds if $i < 0$. By Lemma 10, $q \in \textsc{Corridor}_i$, and by Lemma 11, q has a path π_1 to p_i, the rightmost point in $\textsc{Corridor}_i$ that lies entirely in $\textsc{Corridor}_i$. By Condition (ii) of Lemma 2, there is a point $q' \in \textsc{Strip}_i \cap \textsc{Strip}_{i-1} \cap P$. By Lemma 10, $q' \in \textsc{Corridor}_i$, and by Lemma 11, there is a path π_2 between q' and p_i. Since $q' \in \textsc{Strip}_{i-1}$, q' lies in the same connected component as p in $G[\textsc{Slab}(R,p)]$, and hence there is a path π' from q' to p in $G[\textsc{Slab}(R,p)]$. Concatenating π_1, π_2 and π' we obtain a path π from q to p that lies in $\textsc{Slab}(R,p)$. ◀

We now argue that if p is the rightmost point in a rectangle R, then $\textsc{Piece}(R,p)$ consists of a single slab.

▶ **Lemma 14.** *If p is the last point in R according to the x-coordinates of the points, then $\textsc{Piece}(R,p)$ consists of a single slab.*

Proof. Assume for the sake of contradiction that $\textsc{Upb}(R,p)$ exists. By definition of $\textsc{Upb}(R,p)$, there are two points $a, b \in \textsc{Upb}(R,p)$, such that $x(a) < x_-(R) < x(p) < x_+(R) < x(b)$, as p is the last point in R. But this implies $\textsc{Upb}(R,p)$ is pierced by R, a contradiction. Therefore, $\textsc{Upb}(R,p)$ does not exist. Similarly, $\textsc{Lpb}(R,p)$ does not exist, and hence $\textsc{Piece}(R,p)$ consists of a single slab. ◀

▶ **Theorem 15.** *Algorithm 1 constructs a planar support.*

Proof. By construction, the edges of the graph G constructed by Algorithm 1 are valid Delaunay edges of type $\{\urcorner, \lrcorner\}$. To obtain a plane embedding, we replace each edge $e = \{p, q\}$ by the diagonal of the rectangle $R(pq)$ joining p and q. We call these the *diagonal edges*. It is clear that no diagonal edge pierces a rectangle. If two diagonal edges cross, then it is easy to check that either the corresponding edges cross, or they are not Delaunay. For a rectangle $R \in \mathcal{R}$, let p be the last point in R. Lemma 14 implies that there is only one slab, namely R, and Lemma 13 implies $\textsc{Slab}(R,p)$ is connected. Since R was arbitrary, this implies Algorithm 1 constructs a support. ◀

4.2 Implementation

In this section, we show that Algorithm 1 can be implemented to run in $O(n \log^2 n + (m + n) \log m)$ time with appropriate data structures, where $|\mathcal{R}| = m$, and $|P| = n$. At any point in time, our data structure maintains a subset of points that lie to the left of the sweep line ℓ. It also maintains for each rectangle R intersecting ℓ, the interval $[y_-(R), y_+R]$ corresponding to R. When the sweep line arrives at the left side of a rectangle, the corresponding interval is inserted into the data structure. The interval is removed from the data structure when the sweep line arrives at the right side of the rectangle. Similarly, whenever we sweep over a point p, we insert it into the data structure. In addition, we do the following when the sweep line arrives at a point p:

1. Find the upper and lower barriers at p.
2. Query the data structure to find the set Q of points q which *i)* lie to the left of p and between the upper and lower barriers at p (orthogonal range query) so that *ii)* qp is a Delaunay edge.
3. We add the edge qp for every $q \in Q$ to our planar support. For each edge we add, we remove the points in the data structure that are *occluded* by the edges. These are the points whose y-coordinates lie in the range corresponding to the vertical side of the L-shape for qp.

Our data structure is implemented by combining three different existing data structures. For Step 1, we use a balanced binary search tree $\mathcal{T}_1^u$ augmented so that it can answer range minima or maxima queries. For any rectangle R intersecting the sweep line ℓ, let

(y_1, y_2) denote the interval corresponding to the projection of R on the y-axis. $\mathcal{T}_1^u$ stores the key-value pair (y_1, y_2) with y_1 as the key and y_2 as the value. To find the upper barrier at a point $p = (x, y)$ we need to find the smallest value associated with keys that are at least x. If we augment a standard balanced binary search tree so that at each node we also maintain the smallest value associated with the keys in the subtree rooted at that node, such a query takes $O(\log m)$ time. An analogous search tree $\mathcal{T}_1^b$ is used to find the lower barrier at any point.

To implement Step 2, we use a dynamic data structure $\mathcal{T}_2^b$ due to Brodal [10] which maintains a subset of the points to the left of ℓ and can report points in any query rectangle Q that are not dominated by any of the other points in time $O(\log^2 n + k)$ where k is the number of reported points. We say that a point u is dominated by a point v if both x and y coordinates of u are smaller than those of v. The data structure also supports insertions or deletions of points in $O(\log^2 n)$ time. When the sweep line arrives at a point p, we can use $\mathcal{T}_2^b$ to find all points q that lie to the left of p and below p so that the edge qp is a Delaunay edge (as qp of shape ⌟ is Delaunay iff there is no other point in the range below and to the left of p that dominates q). An analogous data structure $\mathcal{T}_2^u$ is used to find the points q which lie above and to the left of p so that qp (of shape ⌝) is Delaunay.

To implement Step 3, we use a dynamic 1D range search data structure $\mathcal{T}_3$ which also stores a subset of the points to the left of ℓ, supports insertions and deletions in $O(\log n)$ time and can report in $O(\log n + k)$ time the subset of stored points that lie in a given range of y-coordinates (corresponding the vertical side of each added edge), where k is the number of points reported. The points identified are removed from $\mathcal{T}_2^u, \mathcal{T}_2^b$ and $\mathcal{T}_3$.

By the correctness of Algorithm 1 proved in Section 4.1, at any point in time, the current graph is a support for the set of rectangles that lie completely to the left of the sweep line. Thus, if the sweep line ℓ is currently at a point p and q is a point to the left of ℓ, the only rectangles that qp may discretely pierce are those that intersect ℓ. A simple but important observation is that if qp is Delaunay then qp pierces a rectangle iff the vertical portion of L-shape forming the edge pq pierces the rectangle. To see this note that the horizontal portion of the L-shape cannot pierce any rectangle since such a rectangle would not intersect ℓ. The L-shape also cannot (discretely) pierce a rectangle containing the corner of the L-shape since then the edge qp would not be a Delaunay edge. Thus, in order to avoid edges that pierce other rectangles, it suffices to restrict q to lie between the upper and lower barriers at p. Thus Step 1 above ensures that edges found in Step 2 don't pierce any of the rectangles. Similarly, Step 3 ensures that the edges we add in Step 2 don't intersect previously added edges.

The overall time taken by the data structures used by Step 1 is $O((m + n) \log m)$ since it takes $O(\log m)$ time to insert or delete the key-value pair corresponding to any of the m rectangles, and it takes $O(\log m)$ time to query the data structure for the upper and lower barriers at any of the n points. The overall time taken by the data structure in Step 2 is $O(n \log^2 n)$ since there are most $O(n)$ insert, delete, and query operations, and the total number of points reported in all the queries together is $O(n)$. The overall time taken by the data structure in Step 3 is $O(n \log n)$ we only add $O(n)$ edges in the algorithm and the query corresponding to each edge takes $O(\log n)$ time. Each of the reported points is removed from the data structure but since each point is removed only once, the overall time for such removals is also $O(n \log n)$. The overall running time of our algorithm is therefore $O(n \log^2 n + (m + n) \log m)$.

References

1 Emmanuelle Anceaume, Maria Gradinariu, Ajoy Kumar Datta, Gwendal Simon, and Antonino Virgillito. A semantic overlay for self-peer-to-peer publish/subscribe. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 22–22. IEEE, 2006.

2 Daniel Antunes, Claire Mathieu, and Nabil H. Mustafa. Combinatorics of local search: An optimal 4-local Hall's theorem for planar graphs. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 8:1–8:13, 2017. doi:10.4230/LIPICS.ESA.2017.8.

3 Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 2–13, 2007.

4 Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, and Antonino Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA. *The Computer Journal*, 50(4):444–459, 2007. doi:10.1093/COMJNL/BXM002.

5 Aniket Basu Roy, Sathish Govindarajan, Rajiv Raman, and Saurabh Ray. Packing and covering with non-piercing regions. *Discrete & Computational Geometry*, 2018.

6 Sergey Bereg, Krzysztof Fleszar, Philipp Kindermann, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff. Colored non-crossing Euclidean Steiner forest. In *Algorithms and Computation: 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 429–441. Springer, 2015. doi:10.1007/978-3-662-48971-0_37.

7 Sergey Bereg, Minghui Jiang, Boting Yang, and Binhai Zhu. On the red/blue spanning tree problem. *Theoretical computer science*, 412(23):2459–2467, 2011. doi:10.1016/J.TCS.2010.10.038.

8 Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. Blocks of hypergraphs: applied to hypergraphs and outerplanarity. In *Combinatorial Algorithms: 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers 21*, pages 201–211. Springer, 2011. doi:10.1007/978-3-642-19222-7_21.

9 Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. Path-based supports for hypergraphs. *Journal of Discrete Algorithms*, 14:248–261, 2012. doi:10.1016/J.JDA.2011.12.009.

10 Gerth Stølting Brodal and Konstantinos Tsakalidis. Dynamic planar range maxima queries. In *International Colloquium on Automata, Languages, and Programming*, pages 256–267. Springer, 2011. doi:10.1007/978-3-642-22006-7_22.

11 Kevin Buchin, Marc J van Kreveld, Henk Meijer, Bettina Speckmann, and KAB Verbeek. On planar supports for hypergraphs. *Journal of Graph Algorithms and Applications*, 15(4):533–549, 2011. doi:10.7155/JGAA.00237.

12 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012. doi:10.1007/S00454-012-9417-5.

13 Raphaël Chand and Pascal Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lisbon, Portugal, August 30-September 2, 2005. Proceedings 11*, pages 1194–1204. Springer, 2005. doi:10.1007/11549468_130.

14 Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 109–118, 2007. doi:10.1145/1281100.1281118.

15 Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In *Proceedings of the Thirty-first International Symposium on Computational Geometry*, SoCG '15, pages 329–343, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.SOCG.2015.329.

16 Ding-Zhu Du. An optimization problem on graphs. *Discrete applied mathematics*, 14(1):101–104, 1986. doi:10.1016/0166-218X(86)90010-7.

17 Ding-Zhu Du and Dean F Kelley. On complexity of subset interconnection designs. *Journal of Global Optimization*, 6(2):193–205, 1995. doi:10.1007/BF01096768.

18 Ding-Zhu Du and Zevi Miller. Matroids and subset interconnection design. *SIAM journal on discrete mathematics*, 1(4):416–424, 1988. doi:10.1137/0401042.

19 Frédéric Havet, Dorian Mazauric, Viet-Ha Nguyen, and Rémi Watrigant. Overlaying a hypergraph with a graph with bounded maximum degree. *Discrete Applied Mathematics*, 319:394–406, 2022. doi:10.1016/J.DAM.2022.05.022.

20 Jun Hosoda, Juraj Hromkovič, Taisuke Izumi, Hirotaka Ono, Monika Steinová, and Koichi Wada. On the approximability and hardness of minimum topic connected overlay and its special instances. *Theoretical Computer Science*, 429:144–154, 2012. doi:10.1016/J.TCS.2011.12.033.

21 Ferran Hurtado, Matias Korman, Marc van Kreveld, Maarten Löffler, Vera Sacristán, Akiyoshi Shioura, Rodrigo I Silveira, Bettina Speckmann, and Takeshi Tokuyama. Colored spanning graphs for set visualization. *Computational Geometry*, 68:262–276, 2018. doi:10.1016/J.COMGEO.2017.06.006.

22 David S Johnson and Henry O Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *Journal of graph theory*, 11(3):309–325, 1987. doi:10.1002/JGT.3190110306.

23 Ephraim Korach and Michal Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98:385–414, 2003. doi:10.1007/S10107-003-0410-X.

24 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi Varadarajan. Guarding terrains via local search. *Journal of Computational Geometry*, 5(1):168–178, 2014. doi:10.20382/JOCG.V5I1A9.

25 Nabil H Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/S00454-010-9285-9.

26 Melih Onus and Andréa W Richa. Minimum maximum-degree publish–subscribe overlay network design. *IEEE/ACM Transactions on Networking*, 19(5):1331–1343, 2011. doi:10.1109/TNET.2011.2144999.

27 Rajiv Raman and Saurabh Ray. Constructing planar support for non-piercing regions. *Discrete & Computational Geometry*, 64(3):1098–1122, 2020. doi:10.1007/S00454-020-00216-W.

28 Rajiv Raman and Saurabh Ray. On the geometric set multicover problem. *Discret. Comput. Geom.*, 68(2):566–591, 2022. doi:10.1007/s00454-022-00402-y.

29 Rajiv Raman and Karamjeet Singh. On hypergraph supports, 2024. arXiv:2303.16515.

30 AA Voloshina and VZ Feinberg. Planarity of hypergraphs. In *Doklady Akademii Nauk Belarusi*, volume 28, pages 309–311. Akademii Nauk Belarusi F Scorina Pr 66, room 403, Minsk, Byelarus 220072, 1984.

31 TRS Walsh. Hypermaps versus bipartite maps. *Journal of Combinatorial Theory, Series B*, 18(2):155–163, 1975.

32 Alexander Aleksandrovich Zykov. Hypergraphs. *Russian Mathematical Surveys*, 29(6):89, 1974.

A Dichotomy Theorem for Linear Time Homomorphism Orbit Counting in Bounded Degeneracy Graphs

Daniel Paul-Pena
University of California, Santa Cruz, CA, USA

C. Seshadhri
University of California, Santa Cruz, CA, USA

Abstract

Counting the number of homomorphisms of a pattern graph H in a large input graph G is a fundamental problem in computer science. In many applications in databases, bioinformatics, and network science, we need more than just the total count. We wish to compute, for each vertex v of G, the number of H-homomorphisms that v participates in. This problem is referred to as *homomorphism orbit counting*, as it relates to the orbits of vertices of H under its automorphisms.

Given the need for fast algorithms for this problem, we study when near-linear time algorithms are possible. A natural restriction is to assume that the input graph G has bounded degeneracy, a commonly observed property in modern massive networks. Can we characterize the patterns H for which homomorphism orbit counting can be done in near-linear time?

We discover a dichotomy theorem that resolves this problem. For pattern H, let ℓ be the length of the longest induced path between any two vertices of the same orbit (under the automorphisms of H). If $\ell \leq 5$, then H-homomorphism orbit counting can be done in near-linear time for bounded degeneracy graphs. If $\ell > 5$, then (assuming fine-grained complexity conjectures) there is no near-linear time algorithm for this problem. We build on existing work on dichotomy theorems for counting the total H-homomorphism count. Surprisingly, there exist (and we characterize) patterns H for which the total homomorphism count can be computed in near-linear time, but the corresponding orbit counting problem cannot be done in near-linear time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Homomorphism counting, Bounded degeneracy graphs, Fine-grained complexity, Orbit counting, Subgraph counting

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.54

Related Version *Full Version*: `https://arxiv.org/abs/2211.08605`

Funding Both authors are supported by NSF CCF-1740850, CCF-1839317, CCF-2402572, and DMS-2023495.

1 Introduction

Analyzing the occurrences of a small pattern graph H in a large input graph G is a central problem in computer science. The theoretical study has led to a rich and immensely deep theory [39, 20, 30, 24, 40, 2, 23, 45, 51, 17, 16]. The applications of graph pattern counts occur across numerous scientific areas, including logic, biology, statistical physics, database theory, social sciences, machine learning, and network science [34, 19, 22, 18, 27, 13, 29, 42, 59, 45, 25, 44]. (Refer to the tutorial [53] for more details on applications.)

A common formalism used for graph pattern counting is *homomorphism counting*. The pattern graph is denoted $H = (V(H), E(H))$ and is assumed to have constant size. The input graph is denoted $G = (V(G), E(G))$. Both graphs are simple and do not contain

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 54; pp. 54:1–54:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

self-loops. An H-homomorphism is a map $f : V(H) \to V(G)$ that preserves edges. Formally, $\forall(u, v) \in E(H)$, $(f(u), f(v)) \in E(G)$. Let $\mathrm{Hom}_H(G)$ denote the number of distinct H-homomorphisms in G.

Given the importance of graph homomorphism counts, the study of efficient algorithms for this problem is a subfield in itself [35, 3, 18, 27, 26, 24, 13, 23, 14, 51]. The simplest version of this problem is when H is a triangle, itself a problem that attracts much attention. Let $n = |V(G)|$ and $k = |V(H)|$. Computing $\mathrm{Hom}_H(G)$ is $\#W[1]$-hard when parameterized by k (even when H is a k-clique), so we do not expect $n^{o(k)}$ algorithms for general H [24]. Much of the algorithmic study of homomorphism counting is in understanding conditions on H and G when the trivial n^k running time bound can be beaten.

Our work is inspired by the challenges of modern applications of homomorphism counting, especially in network science. Typically, n is extremely large, and only near-linear time ($n \cdot \mathrm{poly}(\log n)$) algorithms are feasible. Inspired by a long history and recent theory on this topic, we focus on *bounded degeneracy* input graphs (we say bounded degeneracy graphs to refer to graphs belonging to classes of graphs with bounded degeneracy). This includes all non-trivial minor-closed graph families, such as planar graphs, bounded genus graphs, and bounded tree-width graphs. Many practical algorithms for large-scale graph pattern counting use algorithms for bounded degeneracy graphs [2, 38, 45, 43, 37, 44]. Real-world graphs typically have a small degeneracy, comparable to their average degree ([32, 37, 55, 5, 9], also Table 2 in [5]).

Secondly, many modern applications for homomorphism counting require more fine-grained statistics than just the global count $\mathrm{Hom}_H(G)$. The aim is to find, *for every vertex v of G*, the number of homomorphisms that v participates in. Seminal work in network analysis for bioinformatics plots the distributions of these per-vertex counts to compare graphs [36, 46]. Orbit counts can be used to generate features for vertices, sometimes called the graphlet kernel [54]. In the past few years, there have been many applications of these per-vertex counts [10, 59, 52, 57, 4, 58, 50, 60, 61].

Algorithms for this problem require considering the "roles" that v could play in a homomorphism. For example, in a 7-path (a path of length 6) there are 4 different roles: a vertex v could be in the middle, could be at the end, or at two other positions. These roles are colored in Fig. 1. The roles are called *orbits* (defined in the Section 3), and the problem of *H-homomorphism orbit counting* is as follows: for every orbit ψ in H and every vertex v in G, output the number of homomorphisms of H where v participates in the orbit ψ. This is the main question addressed by our work:

> *What are the pattern graphs H for which the H-homomorphism orbit counting problem is computable in near-linear time (when G has bounded degeneracy)?*

Recent work of Bressan followed by Bera-Pashanasangi-Seshadhri introduced the question of homomorphism counting for bounded degeneracy graphs, from a fine-grained complexity perspective [14, 8]. A dichotomy theorem for near-linear time counting of $\mathrm{Hom}_H(G)$ was provided in subsequent work [6]. Assuming fine-grained complexity conjectures, $\mathrm{Hom}_H(G)$ can be computed in near-linear time iff the longest induced cycle of H has length at most 5. It is natural to ask whether these results extend to orbit counting.

1.1 Main Result

We begin with some preliminaries. The input graph $G = (V(G), E(G))$ has n vertices and m edges. A central notion in our work is that of graph degeneracy, also called the coloring number.

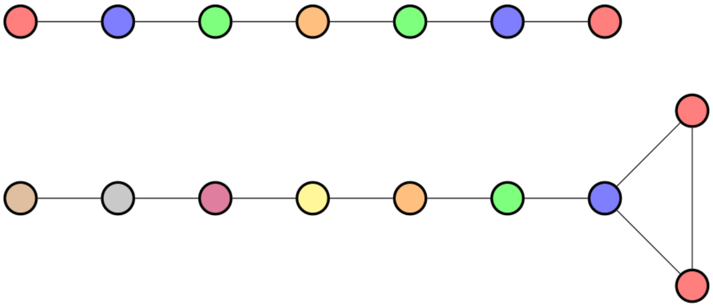


Figure 1 Examples of orbits and LIPCO values. Vertices in the same orbit have the same color. The top graph is the 7-path (a path of length 6). There is an induced path of length 6 between the red vertices, hence the LIPCO of this graph is 6. Theorem 5 implies that we can not compute OrbitHom in near-linear time.
The bottom graph adds a triangle at the end, breaking the symmetry, and the only vertices in the same orbit in that graph are the red ones. The LIPCO in this graph is now less than 6 so we can compute OrbitHom in near-linear time.

▶ **Definition 1.** *A graph G is κ-degenerate if the minimum degree in every subgraph of G is at most κ.*

The degeneracy of G is the minimum value of κ such that G is κ-degenerate.

A family of graphs has *bounded degeneracy* if the degeneracy is constant with respect to the graph size. Bounded degeneracy graph classes are extremely rich. For example, all non-trivial minor-closed families have bounded degeneracy. This includes bounded treewidth graphs. Preferential attachment graphs also have bounded degeneracy; real-world graphs have a small value of degeneracy (often in the 10s) with respect to their size (often in the hundreds of millions) [5].

We assume the pattern graph $H = (V(H), E(H))$ to have a constant number of vertices. (So we suppress any dependencies on purely the size of H.) Consider the group of automorphisms of H. The vertices of H can be partitioned into *orbits*, which consist of vertices that can be mapped to each other by some automorphism (defined formally in Definition 6). For example, in Fig. 1, the 7-path has four different orbits, where each orbit has the same color. The 7-path with a hanging triangle (in Fig. 1) has more orbits, since the pattern is no longer symmetric with respect to the "center" of the 7-path and hence the opposite "ends" of the 7-path cannot be mapped by a non-trivial automorphism.

The set of orbits of the pattern H is denoted $\Psi(H)$. Let $\Phi(H, G)$ be the set of homomorphisms from H to G ($\mathrm{Hom}_H(G) = |\Phi(H, G)|$). We now define our main problem.

▶ **Definition 2.** *Homomorphism Orbit Counts: For each orbit $\psi \in \Psi(H)$ and vertex $v \in V(G)$, define $\mathrm{OrbitHom}_{H,\psi}(v)$ to be the number of H-homomorphisms mapping a vertex of ψ to v. Formally, $\mathrm{OrbitHom}_{H,\psi}(v) = |\{\phi \in \Phi(H, G) \colon \exists h \in \psi,\ \phi(h) = v\}|$.*

*The problem of H-*homomorphism orbit counting *is to output the values $\mathrm{OrbitHom}_{H,\psi}(v)$ for all $v \in V(G), \psi \in \Psi(H)$. (Abusing notation, $\mathrm{OrbitHom}_H(G)$ refers to the list/vector of all of these values.)*

Note that for a given H, the size of the output is $n|\Psi(H)|$ (recall $n = |V(G)|$). For example, when H is the 7-path, we will get $4n$ counts, for each vertex and each of the four orbits.

Our main result is a dichotomy theorem that precisely characterizes patterns H for which $\mathrm{OrbitHom}_H(G)$ can be computed in near-linear time. We introduce a key definition.

▶ **Definition 3.** *For a pattern H, the* Longest Induced Path Connecting Orbits *of H, denoted $LIPCO(H)$ is defined as follows. It is the length of the longest induced simple path, measured in edges, between any two vertices h, h' in H (where h may be equal to h', forming a cycle) in the same orbit.*

Again refer to Fig. 1. The 7-path has a LIPCO of six, since the ends are in the same orbit. On the other hand, the second pattern (7-path with a triangle) has a LIPCO of 3 due to the triangle.

The Triangle Detection Conjecture was introduced by Abboud and Williams on the complexity of determining whether a graph has a triangle [1]. It is believed that this problem cannot be solved in near-linear time, and indeed, may even require $\Omega(m^{4/3})$ time. We use this conjecture for the lower bound of our main theorem.

▶ **Conjecture 4** (Triangle Detection Conjecture [1]). *There exists a constant $\gamma > 0$ such that in the word RAM model of $O(\log n)$ bits, any algorithm to detect whether an input graph on m edges has a triangle requires $\Omega(m^{1+\gamma})$ time in expectation.*

Our main theorem proves that the LIPCO determines the dichotomy. Note that because G is a bounded degeneracy graph we have $m = O(n)$, we will be expressing the bounds in terms of m.

▶ **Theorem 5** (Main Theorem). *Let G be a graph with n vertices, m edges, and bounded degeneracy. Let $\gamma > 0$ denote the constant from the Triangle Detection Conjecture (Conjecture 4).*

- *If $LIPCO(H) \leq 5$: there exists a deterministic algorithm that computes* $\mathrm{OrbitHom}_H(G)$ *in time $O(m \log n)$.*[1]
- *If $LIPCO(H) > 5$: assume the Triangle Detection Conjecture. There is no algorithm with (expected) running time $O(m^{1+\gamma})$ that computes* $\mathrm{OrbitHom}_H(G)$.

Orbit Counting vs Total Homomorphism Counting

In the following discussion, we use "linear" to actually mean near-linear, we assume that the Triangle Detection Conjecture is true, and we assume that G has bounded degeneracy.

One of the most intriguing aspects of the dichotomy of Theorem 5 is that it differs from the condition for getting the total homomorphism count. As mentioned earlier, the inspiration for Theorem 5 is the analogous result for determining $\mathrm{Hom}_H(G)$. There is a near-linear time algorithm iff the length of the longest induced cycle (LICL) of H is at most five. Since the definition of LIPCO considers induced cycles (induced path between a vertex to itself), if $LIPCO(H) \leq 5$, then $LICL(H) \leq 5$. This implies, not surprisingly, that the total homomorphism counting problem is easier than the orbit counting problem.

But there exist patterns H for which the orbit counting problem is provably harder than total homomorphism counting, a simple example is the 7-path (path with 7 vertices). There is a simple linear time dynamic program for counting the homomorphism of paths. But the endpoints are in same orbit, so the LIPCO is six, and Theorem 5 proves the non-existence of linear time algorithms for orbit counting. On the other hand, the LIPCO of the 6-path is five, so orbit counting can be done in linear time.

Consider the pattern at the bottom of Fig. 1. The LICL is three, so the total homomorphism count can be determined in linear time. Because the ends of the underlying 7-path lie in different orbits, the LIPCO is also three (by the triangle). Theorem 5 provides a linear time algorithm for orbit counting.

[1] The exact dependency on the degeneracy κ of the input graph G is $O\left(\kappa^{|H|-1}\right)$.

1.2 Main Ideas

The starting point for homomorphism counting on bounded degeneracy graphs is the seminal work of Chiba-Nishizeki on using acyclic graph orientations [20]. It is known that, in linear time, the edges of a bounded degeneracy graph can be acyclically oriented while keeping the outdegree bounded [41]. For clique counting, we can now use a brute force algorithm in all out neighborhoods, and get a linear time algorithm. Over the past decade, various researchers observed that this technique can generalize to certain other pattern graphs [21, 45, 43, 44]. Given a pattern H, one can add the homomorphism counts of all acyclic orientations of H for an acyclic orientation of G. In certain circumstances, each acyclic orientation can be efficiently counted by a carefully tailored dynamic program that breaks the oriented H into subgraphs spanned by rooted, directed trees.

Bressan gave a unified treatment of this approach through the notion of *DAG-tree decompositions.* [14] These decompositions give a systematic way of breaking up an oriented pattern into smaller pieces, such that homomorphism counts can be computed by a dynamic program. Bera et al. showed that if the LICL of H is at most 5, then the DAG-treewidth of H is at most one [8, 6]. This immediately implies Bressan's algorithm runs in linear time.

Our result on orbit counting digs deeper into the mechanics of Bressan's algorithm. To run in linear time, Bressan's algorithm requires "compressed" data structures that store information about homomorphism counts. For example, the DAG-tree decomposition based algorithm can count 4-cycles in linear time for bounded degeneracy graphs (this was known from Chiba-Nishizeki as well [20]). But there could exist quadratically many 4-cycles in such a graph. Consider two vertices connected by $\Theta(n)$ disjoint paths of length 2; each pair of paths yields a distinct 4-cycle. Any linear time algorithm for 4-cycle counting has to carefully index directed paths and combine these counts, without actually touching every 4-cycle.

By carefully looking at Bressan's algorithm, we discover that "local" per-vertex information about H-homomorphisms can be computed. Using the DAG-tree decomposition, one can combine these counts into a quantity that looks like orbit counts. Unfortunately, we cannot get exact orbit counts, but rather a weighted sum of homomorphisms.

To extract exact orbit counts, we dig deeper into the relationship between orbit counts and per-vertex homomorphism counts. This requires looking into the behavior of independent sets in the orbits of H. We then design an inclusion-exclusion formula that "inverts" the per-vertex homomorpishm counts into orbit counts. The formula requires orbit counts for other patterns H' that are constructed by merging independent sets in the same orbit of H.

Based on previous results, we can prove that if the LICL of all these H' patterns is at most 5, then $\text{OrbitHom}_H(G)$ can be computed in (near)linear time. This LICL condition over all H' is equivalent to the LIPCO of H being at most 5. Achieving the upper bound of Theorem 5.

The above seemingly ad hoc algorithm optimally characterizes when orbit counting is linear time computable. To prove the matching lower bound, we use tools from the breakthrough work of Curticapean-Dell-Marx [23]. They prove that the complexity of counting linear combinations of homomorphism counts is determined by the hardest individual count (up to polynomial factors). Gishboliner-Levanzov-Shapira give a version of this tool for proving linear time hardness [31]. Consider a pattern H with LIPCO at least six. We can construct a pattern H' with LICL at least six by merging vertices of an orbit in H. We use the tools above to construct a constant number of linear sized graphs $G_1, G_2, \ldots, G_k$ such that a linear combination of H-orbit counts on these graphs yields the total H'-homomorphism count on G. The latter problem is hard by existing bounds, and hence the hardness bounds translate to H-orbit homomorphism counting.

2 Related Work

Homomorphism and subgraph counting on graphs is an immense topic with an extensive literature in theory and practice. For a detailed discussion of practical applications, we refer the reader to a tutorial [53].

Homomorphism counting is intimately connected with the treewidth of the pattern H. The notion of tree decomposition and treewidth were introduced in a seminal work by Robertson and Seymour [47, 48, 49]; although it has been discovered before under different names [11, 33]. A classic result of Dalmau and Jonsson [24] proved that $\mathrm{Hom}_H(G)$ is polynomial time solvable if and only if H has bounded treewidth, otherwise it is $\#W[1]$-complete. Díaz et al [26] gave an algorithm for homomorphism counting with runtime $O(2^k n^{t(H)+1})$ where $t(H)$ is the treewidth of the pattern graph H and k the number of vertices of H.

To improve on these bounds, recent work has focused on restrictions on the input G [51]. A natural restriction is bounded degeneracy, which is a nuanced measure of sparsity introduced by early work of Szekeres-Wilf [56]. Many algorithmic results exploit low degeneracy for faster subgraph counting problems [20, 28, 2, 38, 45, 43, 37, 44].

Pioneering work of Bressan introduced the concept of DAG-treewidth for faster algorithms for homomorphism counting in bounded degeneracy graphs [14]. Bressan gave an algorithm for counting $\mathrm{Hom}_H(G)$ running in time essentially $m^{\tau(H)}$, where τ denotes the DAG-treewidth. The result also proves that (assuming ETH) there is no algorithm running in time $m^{o(\tau(H)/\log \tau(H))}$.

Bera-Pashanasangi-Seshadhri build on Bressan's methods to discover a dichotomy theorem for linear time homomorphism counting in bounded degeneracy graphs [7, 8]. Gishboliner, Levanzov, and Shapira independently proved the same characterization using slightly different methods [31, 6].

We give a short discussion of the Triangle Detection Conjecture. Itai and Rodeh [35] gave the first non-trivial algorithm for the triangle detection and finding problem with $O(m^{3/2})$ runtime. The current best known algorithm runs in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$ [3], where ω is the matrix multiplication exponent. Even for $\omega = 2$, the bound is $m^{4/3}$ and widely believed to be a lower bound. Many classic graph problems have fine-grained complexity hardness based on Triangle Detection Conjecture [1].

Homomorphism or subgraph orbit counts have found significant use in network analysis and machine learning. Przulj introduced the use of graphlet (or orbit count) degree distributions in bioinformatics [46]. The graphlet kernel of Shervashidze-Vishwanathan-Petri-Mehlhorn-Borgwardt uses vertex orbits counts to get embeddings of vertices in a network [54]. Four vertex subgraph and large cycle and clique orbit counts have been used for discovering special kinds of vertices and edges [59, 50, 60, 61]. Orbits counts have been used to design faster algorithms for finding dense subgraphs in practice [10, 52, 57, 4, 58].

3 Preliminaries

We use G to denote the input graph and H to denote the pattern graph, both $G = (V(G), E(G))$ and $H = (V(H), E(H))$ are simple, undirected and connected graphs. We denote $|V(G)|$ and $|E(G)|$ by n and m respectively and $|V(H)|$ by k.

A pattern graph H is divided into orbits, we use the definition from Bondy and Murty (Chapter 1, Section 2 [12]):

▶ **Definition 6.** *Fix a graph $H = (V(H), E(H))$. An automorphism is a bijection $\sigma : V(H) \to V(H)$ such that $(u, v) \in E(H)$ iff $(\sigma(u), \sigma(v)) \in E(H)$. The group of automorphisms of H is denoted* $\mathrm{Aut}(H)$.

Define an equivalence relation on $V(H)$ as follows. We say that $u \sim v$ $(u, v \in V(H))$ iff there exists an automorphism that maps u to v. The equivalence classes of the relation are called orbits.

We refer to the set of orbits in H as $\Psi(H)$ and to individual orbits in $\Psi(H)$ as ψ. Note that every vertex $h \in V(H)$ belongs to exactly one orbit. We can represent an orbit by a canonical (say lexicographically least) vertex in the orbit. Somewhat abusing notation, we can think of the set of orbits as a subset of vertices of H, where each vertex plays a "distinct role" in H. Fig. 1 has examples of different graphs with their separate orbits.

We now define homomorphisms.

▶ **Definition 7.** *An H-*homomorphism *from H to G is a mapping $\phi : V(H) \to V(G)$ such that for all $(u, v) \in E(H)$, $(\phi(u), \phi(v)) \in E(G)$. We refer to the set of homomorphisms from H to G as $\Phi(H, G)$.*

We now define a series of counts.

- $\mathrm{Hom}_H(G)$: This is the count of H-homomorphisms in G. So $\mathrm{Hom}_H(G) = |\Phi(H, G)|$.
- $\mathrm{OrbitHom}_{H,\psi}(v)$: For a vertex $v \in V(G)$, $\mathrm{OrbitHom}_{H,\psi}(v)$ is the number of H-homomorphisms that map any vertex in the orbit ψ to v. Formally, $\mathrm{OrbitHom}_{H,\psi}(v) = |\{\phi \in \Phi(H, G) : \exists u \in \psi,\ \phi(u) = v\}|$.
- $\mathrm{OrbitHom}_{H,\psi}(G), \mathrm{OrbitHom}_H(G)$: We use $\mathrm{OrbitHom}_{H,\psi}(G)$ to denote the list/vector of counts $\{\mathrm{OrbitHom}_{H,\psi}(v)\}$ over all $v \in V(G)$. Similarly, $\mathrm{OrbitHom}_H(G)$ denotes the sequence of lists of counts $\mathrm{OrbitHom}_{H,\psi}(G)$ over all orbits ψ.

Our aim is to compute $\mathrm{OrbitHom}_H(G)$, which are a set of homomorphism counts. We use existing algorithmic machinery to compute homomorphism counts per vertex of H, so part of our analysis will consist of figuring out how to go between these counts. As we will see, this is where the LIPCO parameter makes an appearance.

Acyclic orientations. These are a key algorithmic tool in efficient algorithms for bounded degeneracy graphs. An acyclic orientation of an undirected graph G is a digraph obtained by directing the edges of G such that the digraph is a DAG. We will encapsulate the application of the degeneracy in the following lemma, which holds from a classic result of Matula and Beck [41].

▶ **Lemma 8.** *Suppose G has degeneracy κ. Then, in $O(m + n)$ time, one can compute an acyclic orientation $G^{\rightarrow}$ of G with the following property. The maximum outdegree of $G^{\rightarrow}$ is precisely κ. ($G^{\rightarrow}$ is also called a* degeneracy orientation.*)*

The set of all acyclic orientations of H is denoted $\Sigma(H)$. Our algorithm will enumerate over all such orientations.

Note that all definitions of homomorphisms carry over to DAGs.

3.1 DAG-tree decompositions

A central part of our result is applying intermediate lemmas from an important algorithm of Bressan for homomorphism counting [14]. This subsection gives a technical overview of Bressan's techique of DAG-tree decompositions and related lemmas. Our aim is to state the key lemmas from previous work that can be used as a blackbox.

The setting is as follows. We have an acyclic orientation $G^{\rightarrow}$ and a DAG pattern P (think of P as a member of $\Sigma(H)$; P is an acyclic orientation of H). Bressan's algorithm gives a dynamic programming approach to counting $\Phi(P, G^{\rightarrow})$.

We introduce some notation. We use the standard notion of reachability in digraphs: vertex v is reachable from u if there is a directed path from u to v.

- S: The set of sources in the DAG P.
- $Reach_P(s)$: For source $s \in S$, $Reach_P(s)$ is the set of vertices in P reachable from s.
- $Reach_P(B)$: Let $B \subseteq S$. $Reach_P(B) = \bigcup_{s \in B} Reach_P(s)$.
- $P[B]$: This is the subgraph of P induced by $Reach_P(B)$.

▶ **Definition 9** (DAG-tree decomposition [14]). *Let P be a DAG with source vertices S. A DAG-tree decomposition of P is a tree $T = (\mathcal{B}, \mathcal{E})$ with the following three properties:*

1. *Each node $B \in \mathcal{B}$ (referred to as a "bag" of sources) is a subset of the source vertices S: $B \subseteq S$.*
2. *The union of the nodes in T is the entire set S: $\bigcup_{B \in \mathcal{B}} B = S$.*
3. *For all $B, B_1, B_2 \in \mathcal{B}$, if B lies on the unique path between the nodes B_1 and B_2 in T, then $Reach(B_1) \cap Reach(B_2) \subseteq Reach(B)$.*

▶ **Definition 10.** *Let P be a DAG. For any DAG-tree decomposition T to P, the DAG-treewidth $\tau(T)$ is defined as $\max_{B \in \mathcal{B}} |B|$. The DAG-treewidth of P, denoted $\tau(P)$, is the minimum value of $\tau(T)$ over all DAG-tree decompositions T of P.*

Two important lemmas. We state two critical results from previous work. Both of these are highly non-trivial and technical to prove. We will use them in a black-box manner. The first lemma, by Bera-Pashanasangi-Seshadhri, connects the Largest Induced Cycle Length (LICL) to DAG-treewidth [8].

▶ **Lemma 11** (Theorem 4.1 in [8]). *For a simple graph H: $LICL(H) \leq 5$ iff $\forall P \in \Sigma(H), \tau(P) = 1$.*

The second lemma is an intermediate property of Bressan's subgraph counting algorithm [15]. We begin by defining homomorphism extensions. Think of some directed pattern P that we are trying to count. Fix a (rooted) DAG-tree decomposition T. Let P' be a subgraph of P, P'' be a subgraph of P'. A P'-homomorphism ϕ' *extends* a P''-homomorphism ϕ'' if $\forall v \in V(P'')$, $\phi'(v) = \phi''(v)$. Basically, ϕ' agrees with ϕ'' wherever the latter is defined.

- $\text{ext}(P', G; \phi)$: Let ϕ be a homomorphism from a subgraph of P' to G. Then $\text{ext}(P', G; \phi)$ is the number of P'-homomorphisms extending ϕ.
- $P[\text{down}(B)]$: Let B be a node in the DAG-tree decomposition T of P. The set $\text{down}(B)$ is the union of bags that are descendants of B in T. Furthermore $P[\text{down}(B)]$ is the pattern induced by $Reach(\text{down}(B))$.

A technical lemma in Bressan's result shows that extension counts can be obtained efficiently. We will refer to the procedure in this lemma as "Bressan's algorithm".

▶ **Lemma 12** (Lemma 5 in [15]). *Let $G^{\rightarrow}$ be a digraph with outdegree at most d and P be a DAG with k vertices. Let $T = (\mathcal{B}, \mathcal{E})$ be a DAG-tree decomposition for P, and B any element of $\mathcal{B}$. There is a procedure, that in time $O(|\mathcal{B}| poly(k) d^{k-\tau(T)} n^{\tau(T)} \log n)$, returns a dictionary storing the following values: for every $\phi : P[B] \rightarrow G^{\rightarrow}$, it has $\text{ext}(P[\text{down}(B)], G; \phi)$.*

Let us explain this lemma in words. For any bag B, which is a set of sources in P, consider $P[B]$, which is the subgraph induced by $Reach_P(B)$. For every $P[B]$-homomorphism ϕ, we wish to count the number of extensions to $P[\text{down}(B)]$ (the subgraph induced by vertices of P reachable by any source in any descendant bag of B).

4 Obtaining Vertex-Centric Counts

We define vertex-centric homomorphism counts, which allows us to ignore orbits and symmetries in H. Quite simply, for vertices $h \in V(H)$ and $v \in V(G)$, we count the number of homomorphisms from H to G that map h to v.

▶ **Definition 13.** *Vertex-centric Counts: For each vertex $h \in V(H)$ and vertex $v \in V(G)$, let $\text{VertexHom}_{H,h}(v)$ be the number of H-homomorphisms that map h to v.*

Let $\text{VertexHom}_H(G)$ denote the list of $\text{VertexHom}_{H,h}(v)$ over all $h \in V(H)$ and $v \in V(G)$.

We can show that the vertex-centric counts can be obtained in near-linear time when $LICL(H) \leq 5$:

▶ **Theorem 14.** *There is an algorithm that takes as input a bounded degeneracy graph G and a pattern H with $LICL(H) \leq 5$, and has the following properties. It outputs $\text{VertexHom}_H(G)$ and runs in $O(n \log n)$ time.*

Before proving this theorem we need to introduce two more lemmas. First, we invoke the following lemma from [15]:

▶ **Lemma 15** (Lemma 4 in [15])**.** *Given any $B \subseteq S$, the set of homomorphisms $\Phi(P[b], G^{\rightarrow})$ has size $O(d^{k-|B|}n^{|B|})$ and can be enumerated in time $O(k^2 d^{k-|B|}n^{|B|})$.*

Second, we show how to use the output of Bressan's algorithm to obtain the Vertex-centric counts:

▶ **Lemma 16.** *Let P be a directed pattern on k vertices, $T = (\mathcal{B}, \mathcal{E})$ be a DAG-tree decomposition of P with $\tau(P) = 1$ (All nodes/bags in T are singletons), and $G^{\rightarrow}$ be a directed graph with n vertices and max degree d. Let b be the root of T and h be any vertex in $P[b]$. We can compute $\text{VertexHom}_{P,h}(v)$ in time $O(poly(k)d^{k-1}n \log n)$.*

Proof. The algorithm of Lemma 12 will return a data structure/dictionary that gives the following values. For each $\phi : P[b] \rightarrow G^{\rightarrow}$, it provides $\text{ext}(P[\text{down}(b)], G; \phi)$. Note that b is the root of T. By the properties of a DAG-tree decomposition, $\text{down}(b)$ contains all vertices of P and $P[\text{down}(b)] = P$. Hence, the dictionary gives the values $\text{ext}(P, G^{\rightarrow}; \phi)$, that is, the number of homomorphisms $\phi' : P \rightarrow G^{\rightarrow}$ that extend ϕ.

Let h be a vertex in $P[b]$. We can partition the set of homomorphisms from $P[b]$ to $G^{\rightarrow}$, $\Phi(P[b], G^{\rightarrow})$, into sets $\Phi_{b,h,v}$ defined as follows. For each $v \in V(G)$, $\Phi_{b,h,v} := \{\phi \in \Phi(P[b], G^{\rightarrow}) : \phi(h) = v\}$.

By Lemma 15 we can list all the homomorphisms $\Phi(P[b], G^{\rightarrow})$ in $O(k^2 d^{k-1} n)$ time, by the same lemma we know that $\Phi(P[b], G^{\rightarrow})$ will have size at most $O(d^{k-1}n)$, hence we can iterate over the list of homomorphisms and check the value of $\phi(h)$. We can then express $\text{VertexHom}_P(G^{\rightarrow})$ as follows:

$$\begin{aligned}\text{VertexHom}_{P,h}(v) &= |\{\phi' \in \Phi(P, G^{\rightarrow}) : \phi'(h) = v\}| \\ &= \sum_{\phi \in \Phi(P[b], G^{\rightarrow}) : \phi(h) = v} ext(P, G^{\rightarrow}; \phi) \\ &= \sum_{\phi \in \Phi_{b,h,v} : \phi(h) = v} ext(P, G^{\rightarrow}; \phi)\end{aligned}$$

We can compute all of these values by enumerating all the elements in $\phi \in \Phi_{b,h,v}$ (over all v), and making a dictionary access to get $ext(P, G^{\rightarrow}; \phi)$. The total running time is $O(k^2 d^{k-1} n \log n)$, where $\log n$ is extra overhead of accessing the dictionary.

By Lemma 12, the dictionary construction takes $O(|\mathcal{B}| poly(k)\ d^{k-\tau(T)} n^{\tau(T)} \log n)$ time. Since $\tau(T) = 1$ and $|\mathcal{B}| = O(k)$, we can express the total complexity as $O(poly(k) d^{k-1} n \log n)$. ◀

We can now complete the proof of Theorem 14:

Proof of Theorem 14. The first step of our algorithm is to construct the degeneracy orientation $G^{\rightarrow}$ of G. By Lemma 8, it can be computed in $O(m + n)$ time. Since G has bounded degeneracy, $G^{\rightarrow}$ has bounded outdegree. When orienting G as $G^{\rightarrow}$, each homomorphism from H to G becomes a homomorphism of exactly one of the directed patterns $P \in \Sigma(H)$ to $G^{\rightarrow}$. We can hence compute $\text{VertexHom}_H(G)$ as the sum of $\text{VertexHom}_P(G^{\rightarrow})$ for every acyclic orientation of H. This is given by the following equation:

$$\text{VertexHom}_H(G) = \sum_{P \in \Sigma(H)} \text{VertexHom}_P(G^{\rightarrow}) \qquad (1)$$

Because $LICL(H) \leq 5$, Lemma 11 implies that for all $P \in \Sigma(H)$, $\tau(H) = 1$. There exists a DAG-tree decomposition $T = (\mathcal{B}, \mathcal{E})$ of P with $\tau(T) = 1$. We use the output of Bressan's algorithm to obtain the Vertex-centric counts.

The DAG-tree decomposition T can be arbitrarily rooted at any node b. Moreover, for each $h \in V(P)$, there must exist some source b such that $h \in P[b]$ (meaning, h is reachable from b). So, by rooting T at all possible nodes (singleton bags), we can ensure that h is in $P[b]$. We can apply Lemma 16 to get all counts $\text{VertexHom}_{P,h}(v)$.

We complete the proof by bounding the running time and asserting correctness.

From Lemma 8, we can compute $G^{\rightarrow}$ in $O(m + n)$. Since G has bounded degeneracy, $m = O(n)$ and the outdegree d is bounded. The number of acyclic orientations of H, $|\Sigma(H)|$ is bounded by $O(k!)$. In each iteration, by Lemma 16, we will take $O(poly(k) d^{k-1} n \log n)$. For constant k and constant d, the running time is $O(n \log n)$.

Now we prove the correctness of the algorithm. Consider each $P \in \Sigma(H)$. Let $T = (\mathcal{B}, \mathcal{E})$ be the DAG-tree decomposition of P. For each $b \in \mathcal{B}$, we compute $\text{VertexHom}_{P,h}(G^{\rightarrow})$ for all the vertices in $h \in P[b]$. By looping over each singleton bag b, we update counts for all vertices in P. Hence, we are computing $\text{VertexHom}_P(G^{\rightarrow})$. Finally, we sum over all $P \in \Sigma(H)$, which by Equation 1, gives us $\text{VertexHom}_H(G)$. ◀

5 From Vertex-Centric to Orbit Counts

We now show how to go from vertex-centric to orbit counts, using inclusion-exclusion. Much of our insights are given by the following definitions.

▶ **Definition 17.** *$\mathcal{IS}(\psi)$: Given a pattern graph H, for every orbit $\psi \in \Psi(H)$ we define $\mathcal{IS}(\psi)$ as the collection of all non empty subsets $S \subseteq \psi$, such that S forms an independent set (i.e. there is no edge in $E(H)$ connecting any two vertices in S).*

Formally, $\mathcal{IS}(\psi) = \{S \subseteq \psi, S \neq \emptyset : \forall\ h, h' \in S, (h, h') \notin E(H)\}$.

▶ **Definition 18.** *H_S: For each set $S \in \mathcal{IS}(\psi)$ we define H_S as the graph resulting from merging all the vertices in S into a single new vertex h_S, removing any duplicate edge.*

We state two more tools in our analysis. The first lemma relates the counts obtained in the previous section ($\text{VertexHom}_{H_S}(G)$) to the desired output ($\text{OrbitHom}_H(G)$).

▶ **Lemma 19** (Inclusion-exclusion formula).

$$\mathrm{OrbitHom}_{H,\psi}(v) = \sum_{S \in \mathcal{IS}(\psi)} (-1)^{|S|+1} \mathrm{VertexHom}_{H_S,h_S}(v)$$

In order to prove this lemma, we need to define the Signature of a homomorphisms. Let ϕ be a homomorphism from H to G, we define $\mathrm{Sig}(\phi, \psi, v)$ to be the subset of vertices from the orbit ψ that are mapped to v in ϕ. Formally $\mathrm{Sig}(\phi, \psi, h) = \{h \in \psi : \phi(h) = v\}$.

We prove a series of claims regarding the signature.

▷ Claim 20. The Signature of ϕ from ψ to v, $\mathrm{Sig}(\phi, \psi, v)$, must form an independent set of vertices in $V(H)$, that is, there are no edges in $E(H)$ connecting two vertices in $\mathrm{Sig}(\phi, \psi, v)$.

Proof. We prove by contradiction. Assume that $S = \mathrm{Sig}(\phi, \psi, v)$ is not an Independent Set of vertices of $V(H)$, that means that we have a pair of vertices $h, h' \in S$ such that there is an edge connecting them. But from the definition of signature we have that $\phi(h) = \phi(h') = v$, however this is not a valid homomorphism from H to G as it is not preserving the (h, h') edge. ◁

The next claim allows us to relate the Signature with the Homomorphism Orbit Counts.

▷ Claim 21.

$$\mathrm{OrbitHom}_{H,\psi}(v) = \sum_{S \in \mathcal{IS}(\psi)} |\{\phi \in \Phi(H, G) : S = \mathrm{Sig}(\phi, \psi, v)\}|$$

Proof. From the definition of Homomorphism Orbit Counts we have that $\mathrm{OrbitHom}_{H,\psi}(v) = |\{\phi \in \Phi(H, G) \ : \exists h \in \psi, \ \ \phi(h) = v\}|$. Hence, suffices to show that $|\{\phi \in \Phi(H, G) \ : \exists h \in \psi, \ \ \phi(h) = v\}| = \sum_{S \in \mathcal{IS}(\psi)} |\{\phi \in \Phi(H, G) : S = \mathrm{Sig}(\phi, \psi, v)\}|$.

Let $\phi \in \Phi(H, G)$ be a homomorphism from H to G such that $\exists h \in \psi, \phi(h) = v$. Let $S = \mathrm{Sig}(\phi, \psi, v)$, we know that $S \neq \emptyset$ as h is mapped to v and from Claim 20 we know that it forms an independent set on the vertices of H. Hence $S \in \mathcal{IS}(\psi)$.

To prove the other direction of the equality, suffices to note that if a homomorphism ϕ contributes to the right side of the equation, then its signature S belongs to $\mathcal{IS}(\psi)$, hence there is at least one vertex $h \in V(H)$ that is mapped to v, and thus ϕ contributes to the left side of the equation. ◁

Now, we will relate the Signature with the Vertex-centric Counts:

▷ Claim 22. For each orbit ψ in H and each vertex v in $V(G)$ we have that $\forall \ S \in \mathcal{IS}(\psi)$:

$$|\phi \in \Phi(H, G) : \forall h \in S, \phi(h) = v| = \sum_{\substack{S' \supseteq S \\ S' \in \mathcal{IS}(\psi)}} |\phi : \mathrm{Sig}(\phi, \psi, v) = S'|$$

Proof. If ϕ is mapping all the vertices in S to v, then the Signature of ϕ from ψ to v must be a superset of S, $\mathrm{Sig}(\phi, \psi, v) \supseteq S$. Hence summing over such sets will reach the equality. Note that we can add the restriction of S' belonging to $\mathcal{IS}(\psi)$ as it is implied from Claim 20. ◁

Let $\Phi' = \Phi(H_S, G)$ be the set of homomorphism from H_S to G. When S forms an independent set there is an equivalence between the homomorphisms in Φ' that map h_S to v and the set of homomorphisms in $\Phi(H, G)$ that map all the vertices of S to v. In fact we can prove the following claim:

▷ Claim 23. If S is not empty and form an independent set:

$$|\phi \in \Phi(H,G) : \ \forall\ h \in S\ \phi(h) = v| = \text{VertexHom}_{H_S,h_S}(v)$$

Proof. From the definition of Vertex-centric Counts we have that $\text{VertexHom}_{H_S,h_S}(v) = |\phi' \in \Phi(H_S,G) : \phi(h_S) = v|$. Hence it suffices to show that:

$$|\phi \in \Phi(H,G) : \forall h \in S\ \phi(h) = v| = |\phi' \in \Phi(H_S,G) : \phi(h_S) = v|$$

We do so by proving that there is a bijection between both sets, that is, a one to one correspondence between them. Let $\Phi_S = \{\phi' \in \Phi(H_S,G) : \phi(h_S) = v\}$ and $\Phi'_S = \{\phi \in \Phi(H,G) : \forall h \in S, \phi(h) = v\}$. We show an invertible function $f : \Phi_S \to \Phi'_S$:

- Given a homomorphism $\phi \in \Phi_S$ we obtain $\phi' = f(\phi) \in \Phi'_S$ by setting $\phi'(h) = \phi(h)\ \forall\ h \in H \setminus S$ and $\phi'(h_S) = v$. This is a valid homomorphism as we are mapping all the vertices in H_S to G and we are preserving the edges.
- Given a homomorphism $\phi' \in \Phi'_S$ we obtain $\phi = f'(\phi') \in \Phi_S$ by setting $\phi(h) = \phi'(h)\ \forall\ h \in H \setminus S$ and $\phi(h) = v\ \forall\ h \in S$. Again this is a valid homomorphism as we are mapping all the vertices in H to G and we are still preserving the edges.

Additionally, we have that for all $\phi \in \Phi_S$, $\phi = f'(f(\phi))$, which completes the proof. ◁

We will show one last claim that will be important when deriving the inclusion-exclusion formula:

▷ Claim 24. Given a graph H, for every orbit $\psi \in \Psi(H)$, any subset $S' \in \mathcal{IS}(\psi)$ satisfies:

$$\sum_{\substack{S \subseteq S' \\ S \neq \emptyset}} (-1)^{|S|+1} = 1$$

Proof.

$$\sum_{\substack{S \subseteq S' \\ S \neq \emptyset}} (-1)^{|S|+1} = \sum_{i=1}^{|S'|} \binom{|S'|}{i} (-1)^{i+1}$$

$$= \sum_{i=1}^{|S'|} \left(\binom{|S'|-1}{i-1} + \binom{|S'|-1}{i} \right) (-1)^{i+1} = \binom{|S'-1|}{0} (-1)^2 = 1$$ ◁

We now have all the tools required to prove Lemma 19:

Proof of Lemma 19.

$$\sum_{S \in \mathcal{IS}(\psi)} (-1)^{|S|+1} \text{VertexHom}_{H_S,h_S}(v)$$

$$= \sum_{S \in \mathcal{IS}(\psi)} (-1)^{|S|+1} |\phi \in \Phi(H,G) : \ \forall\ h \in S\ \phi(u) = v| \qquad \text{(Claim 23)}$$

$$= \sum_{S \in \mathcal{IS}(\psi)} (-1)^{|S|+1} \sum_{\substack{S' \supseteq S \\ S' \in \mathcal{IS}(\psi)}} |\phi : \text{Sig}(\phi, \psi, v) = S'| \qquad \text{(Claim 22)}$$

$$= \sum_{S \in \mathcal{IS}(\psi)} \sum_{\substack{S' \supseteq S \\ S' \in \mathcal{IS}(\psi)}} (-1)^{|S|+1} |\phi : \text{Sig}(\phi, \psi, v) = S'| \qquad \text{(Factor in)}$$

$$= \sum_{S' \in \mathcal{IS}(\psi)} \sum_{\substack{S \subseteq S' \\ S \neq \emptyset}} (-1)^{|S|+1} |\phi : \text{Sig}(\phi, \psi, v) = S'| \qquad \text{(Reorder)}$$

$$= \sum_{S' \in \mathcal{IS}(\psi)} |\phi : \text{Sig}(\phi, \psi, v) = S'| \sum_{\substack{S \subseteq S' \\ S \neq \emptyset}} (-1)^{|S|+1} \qquad \text{(Factor out)}$$

$$= \sum_{S' \in \mathcal{IS}(\psi)} |\phi : \text{Sig}(\phi, \psi, v) = S'| \qquad \text{(Claim 24)}$$

$$= \text{OrbitHom}_{H,\psi}(v) \qquad \text{(Claim 21)} \quad ◀$$

The next lemma relates the Longest Induced Path Connecting Orbits (LIPCO) defined in Definition 3 with the $LICL$ of all the graphs H_S, for all $S \in \mathcal{IS}(\psi)$ and all orbits ψ of H.

▶ **Lemma 25.** *For every graph H, $LIPCO(H) \leq 5$ iff $\forall \psi \in \Psi(H), \forall S \in \mathcal{IS}(\psi)$, $LICL(H_S) \leq 5$.*

Proof. First, we show that if $LIPCO(H) > 5$ then $\exists \psi \in \Psi(H), \exists S \in \mathcal{IS}(\psi), LICL(H_S) > 5$. Consider the longest induced path in H with endpoints in the same orbit $\psi \in \Psi(H)$, let h, h' be the two endpoints of the path. We have two cases:

- $h = h'$: In this case the induced path is actually just an induced cycle of length 6 or more in H including the vertex h. For any ψ and for any $S \subseteq \psi$ with $|S| = 1$ we have that $H_S = H$, and hence $LICL(H_S) > 5$.
- $h \neq h'$: In the other case we have that h, h' are distinct vertices. Consider the set $S = \{h, h'\}$, we have that $S \in \mathcal{IS}(\psi)$ as both $h, h' \in \psi$ and there is no edge connecting them (otherwise we would have a longer induced cycle). We form H_S by combining h and h' into a single vertex, the induced path that we had in H becomes then an induced cycle of length at least 6, which implies $LICL(H_S) > 5$.

Now, we prove that if $\exists \psi \in \Psi(H), \exists S \in \mathcal{IS}(\psi), LICL(H_S) > 5$ then $LIPCO(H) > 5$. Let S be the set such that $LICL(H_S) > 5$. Again, we have two cases:

- $|S| = 1$: We have that $H_S = H$ and hence $LICL(H) > 5$, any vertex in that induced cycle induces a path of the same length with such vertex in both ends, which implies $LIPCO(H) > 5$.
- $|S| > 1$: Let h_S be the vertex in H_S obtained by merging the vertices of S in H. Consider the longest induced cycle in H_S, if that cycle does not contain h_S then that same cycle exists in H and $LICL(H) > 5$, which implies $LIPCO(H) > 5$. Otherwise, we can obtain H by splitting h_S back into separate vertices, there will be two distinct vertices $h, h' \in S$ that are in the two ends of an induced path of the same length in H, thus $LIPCO(H) > 5$. ◀

6 Wrapping it up

In this section we complete the proof of the main theorem for the upper bound. We also give Algorithm 1, which summarizes the entire process.

▶ **Theorem 26.** *There is an algorithm that, given a bounded degeneracy graph G and pattern H with $LIPCO(H) \leq 5$, computes* $\text{OrbitHom}_H(G)$ *in time $O(n \log n)$.*

Proof. Because we have that $LIPCO(H) \leq 5$, using Lemma 25 we get that $\forall \psi \in \Psi(H), \forall S \in \mathcal{IS}(\psi), LICL(H_S) \leq 5$. This means, using Theorem 14, that $\forall \psi \in \Psi(H), \forall S \in \mathcal{IS}(\psi)$ we can compute $\text{VertexHom}_{H_S}(G)$ in time $f(k)O(n \log n)$.

Using Lemma 19 we can compute $\text{OrbitHom}_H(G)$ from the individual counts of $\text{VertexHom}_{H_S}(G)$ (as shown in Algorithm 1), we have at most 2^k sets S, hence the total time complexity necessary to compute $\text{OrbitHom}_H(G)$ is $O(n \log n)$. ◀

Algorithm 1 Homomorphism Orbit Counts $\text{OrbitHom}_H(G)$.

```
for each $\psi \in \Psi(H)$ do
    for $S \in \mathcal{IS}(\psi)$ do
        Compute $\text{VertexHom}_{H_S,h_S}(G)$
    end for
    $\text{OrbitHom}_{H,\psi}(G) = \sum_{S \in IS(\psi)} (-1)^{|S|+1} \text{VertexHom}_{H_S,h_S}(v)$
end for
Return $\text{OrbitHom}_H(G)$
```

7 Lower Bound for computing Homomorphism Orbit Counts

In this section we prove the lower bound of Theorem 5. It will be given by the following theorem:

▶ **Theorem 27.** *Let H be a pattern graph on k vertices with $LIPCO(H) > 5$. Assuming the Triangle Detection Conjecture, there exists an absolute constant $\gamma > 0$ such that for any function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, there is no (expected) $f(\kappa, k)O(m^{1+\gamma})$ algorithm for the* OrbitHom *problem, where m and κ are the number of edges and the degeneracy of the input graph, respectively.*

To prove this Theorem we will show how to express the Homomorphism Orbit Counts for some orbit ψ as a linear combination of Homomorphism counts of non-isomorphic graphs H_S for all S in $\mathcal{IS}(\psi)$. Because $LIPCO(H) > 5$ we will have that the $LICL$ of at least one of these graphs is also greater than 5. We will then show that the hardness of computing Orbit counts in the original graph is the same than the hardness of computing the Homomorphisms counts. Finally we use a previous hardness result from [8] to complete the proof.

First, we introduce the following definition:

▶ **Definition 28.** *Given a pattern graph H and an input graph G, for the orbit ψ of H, we define $Agg(H, G, \psi)$ as the sum over every vertex $v \in V(G)$ of homomorphisms that are mapping some vertex in ψ to v, that is:*

$$Agg(H, G, \psi) = \sum_{v \in V(G)} \text{OrbitHom}_{H,\psi}(v)$$

Note that if we can compute $\text{OrbitHom}_{H,\psi}(v)$ for every vertex v in G then we can also compute $Agg(H, G, \psi)$ in additional linear time. Now, we state the following lemma:

▶ **Lemma 29.** *For every pattern graph H and every orbit $\psi \in \Psi(H)$, there is some number $l = l(H)$ such that the following holds. For every graph G there are some graphs $G_1, ..., G_l$, computable in time $O(|V(G)| + |E(G)|)$, such that $|V(G_i)| = O(|V|)$ and $|E(G_i)| = O(|E|)$ for all $i = 1, ..., l$, and such that knowing $Agg(H, G_1, \psi), ..., Agg(H, G_l, \psi)$ allows one to compute $\text{Hom}_{H_S}(G)$ for all $S \in \mathcal{IS}(\psi)$, in time $O(1)$. Furthermore, if G is $O(1)$-degenerate, then so are $G_1, ..., G_l$.*

First, we can relate the Homomorphism Vertex Counts of a vertex $h \in V(H)$ to Homomorphism Counts from H to G, as given in the following claim:

▷ Claim 30. For all $h \in V(H)$:

$$\sum_{v \in V(G)} \text{VertexHom}_{H,h}(v) = \text{Hom}_H(G)$$

Proof.

$$\begin{aligned}
&\sum_{v \in V(G)} \text{VertexHom}_{H,h}(v) \\
&= \sum_{v \in V(G)} |\{\phi \in \Phi(H,G) : \phi(h) = v\}| && \text{(Def. of VertexHom)} \\
&= |\{\phi \in \Phi(H,G) : \phi(h) \in V(G)\}| && \text{(Sum over whole set)} \\
&= |\Phi(H,G)| && (\forall \phi : \phi(u) \in V(G)) \\
&= \text{Hom}_H(G) && \text{(Def. of Hom)}
\end{aligned}$$

◁

We now state the following Lemma from [6]:

▶ **Lemma 31** (Lemma 4.2 from [6]). *Let $H_1, ..., H_l$ be pairwise non-isomorphic graphs and let $c_1, ..., c_l$ be non-zero constants. For every graph G there are graphs $G_1, ..., G_l$, computable in time $O(|V(G)| + |E(G)|)$, such that $|V(G_i)| = O(|V(G)|)$ and $|E(G_i)| = O(|E(G)|)$ for every $i = 1, ..., l$, and such that knowing $b_j := c_1 \cdot \text{Hom}_{H_1}(G_j) + ... + c_l \cdot \text{Hom}_{H_l}(G_j)$ for every $j = 1, ..., l$ allows one to compute $\text{Hom}_{H_1}(G), ..., \text{Hom}_{H_l}(G)$ in time $O(1)$. Furthermore, if G is $O(1)$-degenerate, then so are $G_1, ..., G_l$.*

We will apply the previous lemma in a similar way as it is used the proof of Lemma 4.1 in [6].

Proof of Lemma 29. Let $H_1, ..., H_l$ be an enumeration of all the graphs H_S for all $S \in \mathcal{IS}(\psi)$, up to isomorphism. This means that $H_1, ..., H_l$ are pairwise non-isomorphic and $\{H_1, ..., H_l\} = \{H_S : S \in \mathcal{IS}(\psi)\}$.

Let $f(i) = (-1)^{|S|+1}|\{S \in \mathcal{IS}(\psi) : H_S \text{ is isomorphic to } H_i\}|$ be the number of sets $S \in \mathcal{IS}(\psi)$ such that H_S is isomorphic to H_i, with the sign being $(-1)^{|S|+1}$. Note that all such sets have equal $|S|$ and that the value of $f(i)$ is always non-zero. We will use h_i to denote the vertex of H_i that correspond to the vertices h_S of the graphs H_S that are isomorphic to H_i. We can express $Agg(H, G, \psi)$ as follows:.

$$\begin{aligned}
Agg(H,G,\psi) &= \sum_{v \in V(G)} \text{OrbitHom}_{H,\psi}(v) && \text{(Def. 28)} \\
&= \sum_{v \in V(G)} \sum_{S \in \mathcal{IS}(\psi)} (-1)^{|S|+1} \text{VertexHom}_{H_S,h_S}(v) && \text{(Lemma 19)} \\
&= \sum_{v \in V(G)} \sum_{i=1}^{l} f(i) \text{VertexHom}_{H_i,h_i}(v) && \text{(Def. of } f(i)) \\
&= \sum_{i=1}^{l} f(i) \sum_{v \in V(G)} \text{VertexHom}_{H_i,h_i}(v) && \text{(Reorder)} \\
&= \sum_{i=1}^{l} f(i) \text{Hom}_{H_i}(G) && \text{(Claim 30)}
\end{aligned}$$

Hence, we have that $Agg(H, G, \psi)$ is a linear combination of homomorphism counts of $H_1, ..., H_l$. We can then use Lemma 31 to complete the proof. ◀

Before we prove Theorem 27, we need to state the following theorem from [8], which gives a hardness result on Homomorphism Counting:

▶ **Theorem 32** (Theorem 5.1 from [8]). *Let H be a pattern graph on k vertices with $LICL \geq 6$. Assuming the Triangle Detection Conjecture, there exists an absolute constant γ such that for any function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, there is no (expected) $f(\kappa, k)O(m^{1+\gamma})$ algorithm for the Hom_H problem, where m and κ are the number of edges and the degeneracy of the input graph, respectively.*

We now have all the tools required to proof Theorem 27:

Proof of Theorem 27. We prove by contradiction. Given a graph G and a pattern H with $LIPCO(H) > 5$, suppose there exists an algorithm that allows us to compute $\mathrm{OrbitHom}_H(G)$ in time $f(\kappa, k)O(m)$, by Lemma 29 we have the existence of some graphs $G_1, ..., G_l$. We can compute $\mathrm{OrbitHom}_H(G_i)$ for all of these graphs in time $f(\kappa, k)O(m)$ and then aggregate the results into $Agg(H, G_i, \psi)$ for all G_i and all $\psi \in \Psi(H)$. Using Lemma 29, that implies that we can compute $\mathrm{Hom}_{H_S}(G)$ for all $S \in \mathcal{IS}(\psi)$ for all $\psi \in \Psi(H)$ in time $f(\kappa, k)O(m)$.

However, if $LIPCO(H) > 5$ then, by Lemma 25, we have that there exists a $S \subseteq \psi$ for some $\psi \in \Psi(H)$ such that $LICL(H_S) > 5$. From Theorem 32 we know that in that case there is no algorithm that computes $\mathrm{Hom}_{H_S}(G)$ in time $f(\kappa, k)O(m^{1+\gamma})$ for some constant $\gamma > 0$. This is a contradiction, and hence no algorithm can compute $\mathrm{OrbitHom}_H(G)$ in $f(\kappa, k)O(m)$ time. ◀

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science*, 2014. doi:10.1109/FOCS.2014.53.

2 Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. In *Proceedings, SIAM International Conference on Data Mining (ICDM)*, 2015. doi:10.1109/ICDM.2015.141.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

4 A. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016. doi:10.1126/science.aad9029.

5 Suman K Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *International Colloquium on Automata, Languages and Programming*, 2020. doi:10.4230/LIPIcs.ICALP.2020.11.

6 Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *Journal of the ACM (JACM)*, 69(3), 2022. doi:10.1145/3520240.

7 Suman K Bera, Noujan Pashanasangi, and C Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *Proc. 11th Conference on Innovations in Theoretical Computer Science*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.38.

8 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2315–2332, 2021. doi:10.1137/1.9781611976465.138.

9 Suman K Bera and C Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Principles of Database Systems*, pages 457–467, 2020. doi:10.1145/3375395.3387665.

10 Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E*, 83:056119, 2011. doi:10.1103/PhysRevE.83.056119.

11 Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973. doi:10.1016/0097-3165(73)90016-2.

12 J.A. Bondy and U.S.R Murty. *Graph Theory*, volume 244. Springer, 2008. doi:10.1007/978-1-84628-970-5.

13 Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006. doi:10.1007/3-540-33700-8_18.

14 Marco Bressan. Faster subgraph counting in sparse graphs. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.6.

15 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021. doi:10.1007/s00453-021-00811-0.

16 Marco Bressan, Leslie Ann Goldberg, Kitty Meeks, and Marc Roth. Counting subgraphs in somewhere dense graphs. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, pages 27:1–27:14, 2023. doi:10.4230/LIPIcs.ITCS.2023.27.

17 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.

18 Graham R Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of combinatorial theory, series B*, 77(2):221–262, 1999. doi:10.1006/jctb.1999.1899.

19 Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. 9th Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977. doi:10.1145/800105.803397.

20 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing (SICOMP)*, 14(1):210–223, 1985. doi:10.1137/0214017.

21 Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009. doi:10.1109/MCSE.2009.120.

22 J. Coleman. Social capital in the creation of human capital. *American Journal of Sociology*, 94:S95–S120, 1988. doi:10.1086/228943.

23 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017. doi:10.1145/3055399.3055502.

24 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

25 Holger Dell, Marc Roth, and Philip Wellnitz. Counting answers to existential questions. In *Proc. 46th International Colloquium on Automata, Languages and Programming*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.113.

26 Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Counting h-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002. doi:10.1016/S0304-3975(02)00017-8.

27 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000. doi:10.1002/1098-2418(200010/12)17:3/4\%3C260::AID-RSA5\%3E3.0.CO;2-W.

28 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.

29 G. Fagiolo. Clustering in complex directed networks. *Phys. Rev. E*, 2007. doi:10.1103/PhysRevE.76.026107.

30 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing (SICOMP)*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

31 Lior Gishboliner, Yevgeny Levanzov, and Asaf Shapira. Counting subgraphs in degenerate graphs, 2020. arXiv:2010.05998, doi:10.48550/arXiv.2010.05998.

32 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006. doi:10.1007/11917496_15.

33 Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8(1-2):171–186, 1976. doi:10.1007/BF01917434.

34 P. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970. doi:10.1016/B978-0-12-442450-0.50028-6.

35 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi:10.1137/0207033.

36 Shalev Itzkovitz, Reuven Levitt, Nadav Kashtan, Ron Milo, Michael Itzkovitz, and Uri Alon. Coarse-graining and self-dissimilarity of complex networks. *Phys. Rev. E*, 71(016127), January 2005. doi:10.1103/PhysRevE.71.016127.

37 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using Turán's theorem. In *Proceedings, International World Wide Web Conference (WWW)*, pages 441–449, 2017. doi:10.1145/3038912.3052636.

38 Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proc. 24th Proceedings, International World Wide Web Conference (WWW)*, pages 495–505. International World Wide Web Conferences Steering Committee, 2015. doi:10.1145/2736277.2741101.

39 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967. doi:10.1007/BF02280291.

40 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

41 David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983. doi:10.1145/2402.322385.

42 Derek O'Callaghan, Martin Harrigan, Joe Carthy, and Pádraig Cunningham. Identifying discriminating network motifs in youtube spam, 2012. arXiv:1202.5216, doi:10.48550/arXiv.1202.5216.

43 Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science*, 2(1), 2017. doi:10.1007/s41109-017-0027-2.

44 Noujan Pashanasangi and C Seshadhri. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. In *Proc. 13th International Conference on Web Search and Data Mining (WSDM)*, pages 447–455, 2020. doi:10.1145/3336191.3371773.

45 Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1431–1440, 2017. doi:10.1145/3038912.3052597.

46 Natasa Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007. doi:10.1093/bioinformatics/btl301.

47 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

48 Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

49 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

50 Rahmtin Rotabi, Krishna Kamath, Jon M. Kleinberg, and Aneesh Sharma. Detecting strong ties using network motifs. In *Proceedings, International World Wide Web Conference (WWW)*, 2017. doi:10.1145/3041021.3055139.

51 Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2161–2180, 2020. doi:10.1137/1.9781611975994.133.

52 Ahmet Erdem Sariyuce, C. Seshadhri, Ali Pinar, and Umit V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings, International World Wide Web Conference (WWW)*, pages 927–937, 2015. doi:10.1145/2736277.2741640.

53 C. Seshadhri and Srikanta Tirthapura. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *Proceedings, International World Wide Web Conference (WWW)*, 2019. doi:10.1145/3308560.3320092.

54 Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009. URL: http://proceedings.mlr.press/v5/shervashidze09a.html.

55 K. Shin, T. Eliassi-Rad, and C. Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, 2018. doi:10.1007/s10115-017-1077-6.

56 George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968. doi:10.1016/S0021-9800(68)80081-X.

57 Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1122–1132, 2015. doi:10.1145/2736277.2741098.

58 Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1451–1460, 2017. doi:10.1145/3038912.3052653.

59 Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1307–1318, 2013. doi:10.1145/2488388.2488502.

60 Hao Yin, Austin R. Benson, and Jure Leskovec. Higher-order clustering in networks. *Phys. Rev. E*, 97:052306, 2018. doi:10.1103/PhysRevE.97.052306.

61 Hao Yin, Austin R. Benson, and Jure Leskovec. The local closure coefficient: A new perspective on network clustering. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 303–311, 2019. doi:10.1145/3289600.3290991.

Optimal Offline ORAM with Perfect Security via Simple Oblivious Priority Queues

Thore Thießen
University of Münster, Germany

Jan Vahrenhold
University of Münster, Germany

Abstract

Oblivious RAM (ORAM) is a well-researched primitive to hide the memory access pattern of a RAM computation; it has a variety of applications in trusted computing, outsourced storage, and multiparty computation. In this paper, we study the so-called *offline* ORAM in which the sequence of memory access locations to be hidden is known in advance. Apart from their theoretical significance, offline ORAMs can be used to construct efficient oblivious algorithms.

We obtain the first optimal offline ORAM with perfect security from oblivious priority queues via time-forward processing. For this, we present a simple construction of an oblivious priority queue with perfect security. Our construction achieves an asymptotically optimal (amortized) runtime of $\Theta(\log N)$ per operation for a capacity of N elements and is of independent interest.

Building on our construction, we additionally present efficient external-memory instantiations of our oblivious, perfectly-secure construction: For the cache-aware setting, we match the optimal I/O complexity of $\Theta(\frac{1}{B}\log\frac{N}{M})$ per operation (amortized), and for the cache-oblivious setting we achieve a near-optimal I/O complexity of $\mathcal{O}(\frac{1}{B}\log\frac{N}{M}\log\log_M N)$ per operation (amortized).

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Cryptographic protocols

Keywords and phrases offline ORAM, oblivious priority queue, perfect security, external memory algorithm, cache-oblivious algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.55

Related Version *Full Version*: `https://arxiv.org/abs/2409.12021` [31]

Acknowledgements We thank all anonymous reviewers for their constructive comments that helped to improve the presentation.

1 Introduction

Introduced by Goldreich and Ostrovsky [16], *oblivious RAM* (*ORAM*) conceals the memory access pattern of any RAM computation. This prevents the leakage of confidential information when some adversary can observe the pattern of memory accesses. We consider oblivious RAM in the offline setting: This allows an additional pre-processing step on the access pattern while still requiring that the access pattern is hidden from the adversary.

Offline ORAMs can be used to construct efficient oblivious algorithms in situations where at least part of the memory access sequence is either known or can be inferred in advance. As a motivating example, consider the classical Gale–Shapley algorithm for the stable matching problem [15, 27]: In each round of the algorithm, up to n parties make a proposal according to their individual preferences. The preferences must be hidden to maintain obliviousness, and thus the memory access pattern may not depend on them. While it seems that the standard algorithm makes online choices, in fact the preferences and the current matching are known before each round, so the proposals can be determined in advance and an offline ORAM can be used to hide the access pattern in each round.

Many of the previous works on offline (and online) ORAMs focus on statistical and computational security: While optimal offline ORAMs are known for computational and statistical security [4, 29], the same is not true for perfect security. We close this gap and obtain the first (asymptotically) optimal offline ORAM with perfect security. We derive our construction from an oblivious priority queue.

For this, we discuss and analyze a construction of an oblivious priority queue simple enough to be considered part of folklore. In fact, both the construction and its analysis can be used in an undergraduate data structures course as an example of how to construct an efficient oblivious data structure from simple building blocks. Our construction reduces the problem to oblivious partitioning where an optimal oblivious algorithm [4] is known.

1.1 Oblivious Data Structures

Conceptually, (offline) ORAM and oblivious priority queues are *oblivious data structures*. Oblivious data structures provide efficient means to query and modify data while not leaking information, e. g., distribution of the data or the operations performed, via the memory access pattern. There are three main applications:

Outsourced Storage. When storing data externally, oblivious data structures can be used in conjunction with encryption. Encryption alone protects the confidentiality of the data at rest, but performing operations may still leak information about queries or the data itself via the access pattern [20].

Trusted Computing. When computing in trusted execution environments, oblivious data structures safeguard against many memory-related side channel attacks [28].

(Secure) Multiparty Computation. In this setting, actors want to (jointly) compute a function without revealing their respective inputs to each other. Here, oblivious data structures have been used to allow for data structure operations with sublinear runtime [32, 23].

1.1.1 Security Definition

In line with standard assumptions for oblivious algorithms [16], we assume the w-bit word RAM model of computation. Let the random variable $\mathrm{Addr}_{\textsc{Op}(x)}$ with

$$\mathrm{Addr}_{\textsc{Op}(x)} \in (\{0, \dots, 2^w - 1\} \times \{\textsc{Read}, \textsc{Write}\})^* \qquad (1)$$

denote the sequence of *memory probes* for $\textsc{Op}(x)$, i. e., the sequence of memory access locations and memory operations performed by operation $\textsc{Op}$ for input x. Access to a constant number of registers (*private memory*) is excluded from the probe sequence.

For *perfect* oblivious security, we require that all data structure operation sequences of length n produce the same memory access pattern:

▶ **Definition 1** (Obliviousness with Perfect Security). *We say that an (online) data structure $\mathcal{D}_N$ with capacity*[1] *N and operations $\textsc{Op}_1, \dots, \textsc{Op}_m$ is oblivious with perfect security iff, for every two sequences of n operations*

$$X = \langle \textsc{Op}_{i_1}(x_1), \dots, \textsc{Op}_{i_n}(x_n) \rangle \quad \textit{and} \quad Y = \langle \textsc{Op}_{j_1}(y_1), \dots, \textsc{Op}_{j_n}(y_n) \rangle$$

with valid inputs x_k, y_k, the memory probe sequences are identically distributed, i. e.,

$$\langle \mathrm{Addr}_{\textsc{Op}_{i_1}(x_1)}, \dots, \mathrm{Addr}_{\textsc{Op}_{i_n}(x_n)} \rangle \equiv \langle \mathrm{Addr}_{\textsc{Op}_{j_1}(y_1)}, \dots, \mathrm{Addr}_{\textsc{Op}_{j_n}(y_n)} \rangle .$$

[1] To hide the type of operation performed, in particular for intermixed Insert and Delete sequences, it is assumed that the data structure has a fixed capacity N determined a priori. This assumption does not limit any of our analyses, as the capacity can be adjusted using standard (doubling) techniques with (amortized) constant asymptotic overhead per operation.

Table 1 Oblivious priority queues supporting Insert, Min, and DeleteMin. Deletions are noted as supported if an operation Delete, ModifyPriority, or DecreasePriority is available.

security	*runtime*	*priv. memory*	*deletion*	
perfect[p]	$\mathcal{O}(\log^2 N)^{a}$	$\mathcal{O}(1)$	no	[32]
statistical	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(\omega(1) \cdot \log N)$	no	[33]
statistical	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(\omega(1) \cdot \log N)$	yes[r]	[23, Path ORAM variant]
perfect	$\mathcal{O}(\log^2 N)^{a}$	$\mathcal{O}(1)$	no	[26]
statistical	$\mathcal{O}(\log N)^{a}$	$\mathcal{O}(\omega(1) \cdot \log N)$	yes	[22]
statistical	$\mathcal{O}(\omega(1) \cdot \log N)$	$\mathcal{O}(1)$	yes[r]	[29, Circuit variant]
perfect	$\mathcal{O}(\log^2 N)^{a}$	$\mathcal{O}(1)$	yes[r]	[19]
perfect	$\mathcal{O}(\log N)^{a}$	$\mathcal{O}(1)$	no	**new**

[p] reveals the operation   [a] amortized runtime complexity   [r] requires an additional reference

The requirement of identical distribution in the above definition can be relaxed to strictly weaker definitions of security by either allowing a negligible statistical distance of the probe sequences (*statistical security*) or allowing a negligible distinguishing probability by a polynomial-time adversary (*computational security*); see Asharov et al. [4] for more details.

Definition 1 immediately implies that the memory probe sequence is independent of the operation arguments – and, by extension, the data structure contents – as well as the operations performed (*operation-hiding security*). As a technical remark, we note that for perfectly-secure data structure operations with determined outputs, the joint distributions of output and memory probe sequence are also identically distributed. This implies that data structures satisfying Definition 1 are universally composable [4].

1.1.2 Offline ORAM

The (online) ORAM is essentially an oblivious array data structure [24]. By using an ORAM as the main memory, any RAM program can generically be transformed into an oblivious program at the cost of an *overhead* per memory access.

The offline ORAM we are considering here, however, is given the sequence I of access locations in advance. While this allows pre-computations on I, the probe sequence must still hide the operations and indices in I. In anticipation of the offline ORAM construction in Section 3, we take a similar approach as Mitchell and Zimmerman [26] and define an offline ORAM as an *online* oblivious data structure with additional information:

▶ **Definition 2** (Offline ORAM). *An offline ORAM is an oblivious data structure $\mathcal{D}_N$ that maintains an array of length N under an annotated online sequence of read and write operations:*

***Read**(i, τ)* *Return the value stored at index i in the array.*

***Write**(i, v', τ)* *Store the value v' in the array at index i.*

The annotation τ indicates the time-stamp of the next operation accessing index i.

Note that this definition implies that $\mathcal{D}_N$ can also be used in an online manner if the time-stamps τ of the next operation accessing the index i are known. When discussing the offline ORAM construction in Section 3, we show how to use sorting and linear scans to compute the annotations τ from the sequence I of access locations given in advance.

Table 2 Best known overhead bounds for online and offline ORAMs with N memory cells, a constant number of private memory cells, and standard parameters [24].

	perfect security		*statistical security*		*comput. security*	
online	$\Omega(\log N)$	[24]	$\Omega(\log N)$	[24]	$\Omega(\log N)$	[24]
	$\mathcal{O}(\log^3 N/\log\log N)$	[11]	$\mathcal{O}(\log^2 N)$	[10]	$\mathcal{O}(\log N)^{\mathrm{p}}$	[5]
offline	$\Omega(\log N)^{\mathrm{i}}$	[16]	$\Omega(\log N)^{\mathrm{i}}$	[16, 8]	$\Omega(1)$	trivial [8]
	$\mathcal{O}(\log^2 N)^{\mathrm{a}}$	e. g., via [26]	$\mathcal{O}(\omega(1)\cdot\log N)$	[29]	$\mathcal{O}(\log N)^{\mathrm{p}}$	[5]
	$\mathcal{O}(\log N)^{\mathrm{a}}$	**new**				

$^{\mathrm{p}}$ assuming a pseudo-random function family $^{\mathrm{i}}$ assuming indivisibility [8] $^{\mathrm{a}}$ amortized

1.2 Previous Work

Oblivious Priority Queues. Because of their many algorithmic applications, oblivious priority queues have been considered in a number of previous works. We provide an overview of previous oblivious priority queue constructions in Table 1.

Jacob et al. [21] show that a runtime of $\Omega(\log N)$ per operation is necessary for oblivious priority queues. Their lower bound holds even when allowing a constant failure probability and relaxing the obliviousness to statistical or computational security.

The first oblivious priority queue construction due to Toft [32] is perfectly-secure and has an amortized runtime of $\mathcal{O}(\log^2 N)$, but reveals the operation performed and lacks operations to delete or modify arbitrary elements. Subsequent perfectly-secure constructions [26, 19] offer operation-hiding security or support additional operations, but do not improve the suboptimal $\mathcal{O}(\log^2 N)$ runtime. A different line of work considers oblivious priority queues with statistical security. Jafargholi et al. [22] and, subsequently, Shi [29] both present constructions with an optimal $\Theta(\log N)$ runtime. All statistically secure priority queue constructions [33, 23, 22, 29] are randomized; many [33, 23, 29] also rely on tree-based ORAMs (e. g., *Path ORAM* [30] or *Circuit ORAM* [10]) in a non–black-box manner.

Offline ORAMs. Though much of the research focuses on online ORAMs, *offline* ORAMs have been explicitly considered in some previous works [26, 8, 22, 29]. We provide an overview of the best known upper and lower bounds for both online and offline ORAM constructions with perfect, statistical, or computational security in Table 2.

Goldreich and Ostrovsky [16] prove a lower bound on the overhead of $\Omega(\log N)$ for (online) ORAMs with perfect security (assuming indivisibility). This bound also applies to offline ORAMs and constructions with statistical security [8].

There is a generic way to construct offline ORAMs from oblivious priority queues (see Section 3). Via their priority queue construction, Shi [29] obtains an optimal offline ORAM with statistical security for a private memory of constant size. For computational security, the state-of-the-art online ORAM construction [5] is simultaneously the best known offline construction (asymptotically). While the upper bounds for statistical and computational security match the (conjectured) $\Omega(\log N)$ lower bound, prior to our work there remained a gap for perfect security.[2]

[2] Boyle and Naor [8] show how to construct an offline ORAM with overhead $\mathcal{O}(\log N)$: In addition to the access locations, their construction must be given the *operands* of the write operations in advance, i. e., the sequence of values to be written. It thus does not fit our more restrictive Definition 2.

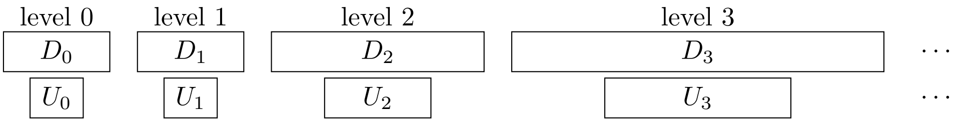


Figure 1 Structure of the oblivious priority queue: Each level $i \in \{0, \ldots, \ell - 1\}$ consists of a down-buffer D_i and an up-buffer U_i half the size of D_i.

1.3 Contributions

Our work provides several contributions to a better understanding of the upper bounds of perfectly-secure oblivious data structures:

- As a main contribution, we present and analyze an oblivious priority queue construction with perfect security. This construction is conceptually simple and achieves the optimal $\Theta(\log N)$ runtime per operation amortized.
  In particular, our construction improves over the previous statistically-secure constructions [22, 29] in that we eliminate the failure probability (perfect security with perfect correctness) and achieve a strictly-logarithmic runtime for $\mathcal{O}(1)$ private memory cells.[3]
- The priority queue implies an optimal $\Theta(\log N)$-overhead offline ORAM with perfect security, closing the gap to statistical and computational security in the offline setting. We show that these bounds hold even for a large number n of operations, i. e., $n = N^{\omega(1)}$.
- We also provide improved external-memory oblivious priority queues: Compared to the I/O-optimal state-of-the-art [22], our cache-aware construction achieves perfect security and only requires a private memory of constant size.
  In the cache-oblivious setting, our construction achieves near-optimal I/O-complexity for perfect security and a private memory of constant size. We are not aware of any previous oblivious priority queues in the cache-oblivious setting.

2 Oblivious Priority Queue from Oblivious Partitioning

An oblivious priority queue data structure maintains up to N elements and must support at least three non-trivial operations prescribed by the abstract data type PriorityQueue:
Insert(k, p). Insert the element $\langle k, p\rangle$ with *priority* p.
Min(). Return the element $\langle k, p_{\min}\rangle$ with the minimal priority $p_{\min}$.
DeleteMin(). Remove the element $\langle k, p_{\min}\rangle$ with minimal priority $p_{\min}$.
We assume that both the key k and the priority p fit in a constant number of memory cells and that the relative order of two priorities p, p' can be determined obliviously in constant time; larger elements introduce an overhead factor in the runtime. To keep the exposition simple, we assume distinct priorities. This assumption can be removed easily, see Section 2.2.

Figure 1 shows the structure of our solution: In a standard data structure layout, it has $\ell \in \Theta(\log N)$ levels of geometrically increasing size. Each level i consists of a *down*-buffer D_i and an *up*-buffer U_i, both of size $\Theta(2^i)$. Insert inserts into the up-buffer U_0 and DeleteMin removes from the down-buffer D_0. Each level i is rebuilt after 2^i operations, moving elements up through the up-buffers and back down through the down-buffers. The main idea guiding the rebuilding is to ensure that all levels $j < i$ can support the next operations until level i is rebuilt; we later formalize this as an invariant for the priority queue (see Lemma 4).

[3] For a private memory of constant size, the construction of Shi [29] requires an additional $\omega(1)$-factor in runtime to achieve a negligible failure probability.

Oblivious Building Blocks. For our construction, we need an algorithm to obliviously permute a given array A of n elements such that the k smallest elements are swapped to the front, followed by the remaining $n - k$ elements. We refer to this problem as *k-selection.*

To obtain an efficient algorithm for k-selection, we use an oblivious modification of the classical (RAM) linear-time selection algorithm [7] as sketched by Lin et al. [25, full version, Appendix E.2]. This reduces k-selection to the *partitioning* problem (also called 1-*bit sorting*). The oblivious k-selection deviates from the classical algorithm in two respects [25]:

- First, it is necessary to ensure that the partitioning step is oblivious. For this, instead of the algorithms proposed by Lin et al., we use the optimal oblivious partitioning algorithm of Asharov et al. [4, Theorem 5.1].[4] This allows us to obtain a linear-time algorithm for k-selection.
- Second, the relative position of the median of medians among the elements cannot be revealed as this would leak information about the input. To address this, Lin et al. propose over-approximating the number of elements and always recursing with approximately $\frac{7n}{10}$ elements.

▶ **Corollary 3** (Oblivious k-Selection via [25, 4]). *There is a deterministic, perfectly-secure oblivious algorithm for the k-selection problem with runtime $\mathcal{O}(n)$ for n elements.*

We describe the algorithm in detail and prove its correctness in the full version [31].

Comparison with Jafargholi et al. [22]. Conceptually, our construction is similar to that of Jafargholi et al. [22]: In both constructions, the priority queue consists of levels of geometrically increasing size with lower-priority elements moving towards the smaller levels. Structuring the construction so that larger levels are rebuilt less frequently is a standard data structure technique to amortize the cost of rebuilding.

The main difference lies in the rebuilding itself: In the construction of Jafargholi et al., level i is split into 2^i nodes; overall, the levels form a binary tree. The elements are then assigned to paths in the tree based on their key [22]. While this allows deleting elements by their key efficiently, this inherently introduces the probability of "overloading" certain nodes, reducing the construction to statistical security (with a negligible failure probability).

We instead use k-selection for rebuilding; this allows us to maintain both perfect correctness and security. Unfortunately, this comes at the cost of a more expensive Delete operation: Since we maintain no order on the keys within each level, it is not possible to efficiently delete arbitrary elements by their key. We note that deleting arbitrary elements is not required for our offline ORAM construction.

2.1 Details of the Construction

The priority queue consists of $\ell := \lceil \log_2 N \rceil$ levels, each with a down-buffer D_i of $2^{\max\{1,i\}}$ elements and an up-buffer U_i of $2^{\max\{0,i-1\}} = \frac{|D_i|}{2}$ elements. An element is a pair $\langle k, p \rangle$ of key k and priority p; each buffer is padded with *dummy* elements to hide the number of "real" elements. Initially, all elements in the priority queue are dummy elements. We refer to a buffer containing only dummy elements as *empty.*

The elements are distributed over the levels via a rebuilding procedure: Level i is rebuilt after exactly 2^i operations. Let Δ_i be the remaining number of operations until level i is rebuilt (with $\Delta_i = 2^i$ initially). After each operation, all counters Δ_i are decremented by

[4] Note that Asharov et al. refer to the partitioning problem as *compaction.* We use the term partitioning to stress that all elements of the input are preserved which is necessary for our definition of k-selection.

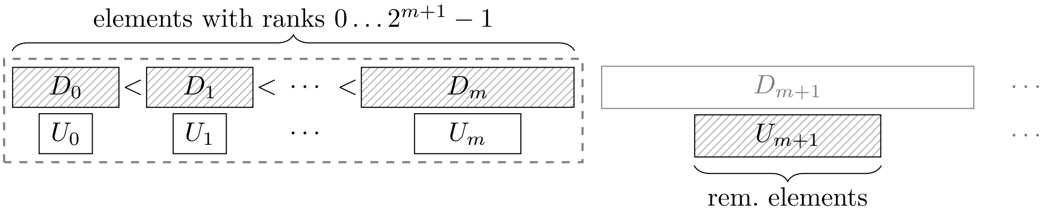

elements with ranks $0 \dots 2^{m+1}-1$
D_0 < D_1 < ⋯ < D_m
U_0
U_1
⋯
U_m
D_{m+1}
U_{m+1}
⋯
rem. elements

Figure 2 Distribution of the elements when rebuilding level m: The up to 2^{m+1} smallest elements in the levels $0, \dots, m$ are distributed over the down-buffers of the first m levels. The up to 2^m remaining elements are inserted into the (empty) up-buffer U_{m+1} of level $m+1$.

one and all levels i with $\Delta_i = 0$ are rebuilt with Rebuild(m) for $m := \max\{i < \ell \mid \Delta_i = 0\}$; note that $\Delta_i = 0$ if and only if $i \le m$. The counter Δ_i of each rebuilt level $i \le m$ is reset to 2^i, so $\Delta_i > 0$ for every level i after each operation.

We will show the correctness of the construction with three invariants (a)–(c):

▶ **Lemma 4** (Invariants). *Before each operation of the priority queue, the following holds:*

(a) *The priority queue contains the correct elements, i. e., $\mathcal{E} = \bigcup_{i<\ell}(U_i \cup D_i)$ where $\mathcal{E}$ denotes the elements that should be contained in the priority queue (with standard semantics).*

(b) *The up-buffer U_0 is empty, i. e., contains exactly one dummy element.*

(c) *For all elements $e := \langle k, p\rangle \in D_i \cup U_i$ with $i \ge 1$, it holds that $\Delta_i \le \operatorname{rank}(e)$ where $\operatorname{rank}(\langle \cdot, p\rangle) := |\{\langle \cdot, p'\rangle \in \mathcal{E} \mid p' < p\}|$ is the (unique) rank of p in the priority queue.*

The most important invariant (c) guarantees that each level $i \ge 1$ is rebuilt before any of its elements are required for Min/DeleteMin in D_0. In turn, this implies that the Δ_i smallest elements potentially required before rebuilding level i are stored in the buffers on levels $0, \dots, i-1$. Formally, this follows since $\Delta_j \le \Delta_i$ for all $j < i$.

For simplicity of exposition, we ignore the details of the rebuilding step for the time being and discuss how the three priority queue operations can be implemented while maintaining the invariants (a) and (c) of Lemma 4:

Insert(k, p). The dummy in U_0 is replaced with the new element $\langle k, p\rangle$. After the operation, the priority queue contains the elements $\mathcal{E}' = \mathcal{E} \cup \{\langle k, p\rangle\} = \bigcup_{i<\ell}(U_i \cup D_i)$. Inserting a new element does not decrease the rank of any element while all counters Δ_i decrease; this implies that invariant (c) is maintained.

Min(). The minimal element $e_{\min}$ with $\operatorname{rank}(e_{\min}) = 0$ must be contained in level 0 since $\Delta_i > 0$ for all $i > 0$ before each operation. Since U_0 is empty, $e_{\min}$ is one of the two elements in D_0. After the operation, the elements $\mathcal{E}' = \mathcal{E} = \bigcup_{i<\ell}(U_i \cup D_i)$ remain the same. Invariant (c) is maintained since the ranks of all elements $e \in \mathcal{E}$ remain unchanged.

DeleteMin(). For this operation, we replace the minimal element $e_{\min} \in D_0$ with a dummy element. After the operation, the priority queue contains the elements $\mathcal{E}' = \mathcal{E} \setminus \{e_{\min}\} = \bigcup_{i<\ell}(U_i \cup D_i)$. Removing the minimum reduces the rank of all other elements by one, but invariant (c) is maintained since all counters Δ_i also decrease.

For the operation-hiding security, we access memory locations for all three operations but only perform updates for the intended operation. For example, DeleteMin will access both U_0 and D_0, but only actually overwrite the minimal element in D_0 with a dummy element. We provide pseudocode for the operations in the full version [31].

We now turn to describing Rebuild(m) (Algorithm 1). As shown in Figure 2, this procedure processes all elements in the levels $0, \dots, m$: The non-dummy elements are distributed into $D_0, \dots, D_m, U_{m+1}$ and the up-buffers $U_0, \dots, U_m$ are emptied, i. e., filled

Algorithm 1 Rebuild the levels $0, \dots, m$ in the oblivious priority queue. Let $A \parallel B$ denote the concatenation of two buffers A and B; $A_{0\dots i}$ denotes the concatenation $A_0 \parallel \cdots \parallel A_i$.

```
1: procedure Rebuild($m$)
2:     KSelect($2^{m+1}, D_{0\dots m} \parallel U_{0\dots m}$)     ▷ move $2^{m+1}$ smallest elements to $D_{0\dots m}$
3:     if $m$ is not the last level then
4:         $U_{m+1} \leftarrow U_{0\dots m}$                        ▷ copy the elements in $U_{0\dots m}$ to $U_{m+1}$
5:         $U_{0\dots m} \leftarrow \langle \bot, \dots, \bot \rangle$      ▷ overwrite $U_{0\dots m}$ with dummy elements
6:     for $i \leftarrow m-1, \dots, 0$ do
7:         KSelect($2^{i+1}, D_{0\dots i+1}$)                 ▷ move $2^{i+1}$ smallest elements to $D_{0\dots i}$
8:     for $i \leftarrow 0, \dots, m$ do
9:         $\Delta_i \leftarrow 2^i$                                ▷ reset counters
```

with dummy elements. The down-buffers $D_0, \dots, D_m$ collectively contain up to 2^{m+1} non-dummy elements. Additionally, the up-buffers $U_0, \dots, U_m$ collectively contain up to 2^m non-dummy elements. All these elements are distributed over the buffers $D_0, \dots, D_m$ and U_{m+1} such that

- D_0 contains the two smallest elements (with ranks 0 and 1),
- the other D_i (for $i \leq m$) each contain the elements with ranks $2^i, \dots, 2^{i+1} - 1$,[5] and
- U_{m+1} contains all remaining elements.

For this, we order the elements by their priority p; dummy elements have no priority and are ordered after non-dummy elements.

We can now prove that the overall construction is correct by showing that Rebuild(m) with $m \coloneqq \max\{i < \ell \mid \Delta_i = 0\}$ maintains the invariants (a)–(c):

Proof of Lemma 4. All invariants trivially hold for the empty priority queue. As described above, the operations Insert, Min, and DeleteMin maintain invariants (a) and (c). After each operation, all counters Δ_i are decremented and the levels i with $\Delta_i = 0$ are rebuilt. We now show that all invariants hold after rebuilding.

Invariants (a) and (b): $\mathcal{E} = \bigcup_{i<\ell}(U_i \cup D_i)$ and U_0 is empty. We first show that prior to Rebuild(m) in each operation where m is not the last level, the up-buffer U_{m+1} is empty. This can be seen by considering the two possible cases:

- If no more than 2^m operation have been performed overall, the level $m+1$ has never been accessed. In this case U_{m+1} is empty since it was empty initially.
- Otherwise, if more than 2^m operations have been performed, the up-buffer U_{m+1} was emptied 2^m operations before by Rebuild(m') for some $m' > m$ (and not accessed since).

This means that by copying the elements in $U_{0\dots m}$ into U_{m+1} (Line 4), only dummy elements are being overwritten and invariant (a) is maintained.

In case m is the last level ($m = \ell - 1$), after Line 2 the up-buffers $U_0, \dots, U_m$ are empty iff there no more than $2^{m+1} = 2^\ell \geq N$ elements in the data structure. This is guaranteed by the capacity bound N. Thus, the up-buffer U_0 is empty after each operation.

[5] Even if there are more than 2^{m+1} elements in the priority queue overall, the buffer D_m may still end up (partially) empty after rebuilding. This is no threat to the correctness, since invariant (c) guarantees that level $m+1$ will be rebuilt before requiring the elements with ranks $\geq 2^m$.

Invariant (c): $\Delta_i \leq \operatorname{rank}(e)$ for all $e \in U_i \cup D_i$ with $i \geq 1$. Next, we show that rebuilding maintains the rank invariant for all redistributed elements. Using k-selections to redistribute the elements makes sure that a buffer in level i receives non-dummy elements only if all $D_0, \ldots, D_{i-1}$ have been filled to capacity. Consider any level $i \geq 1$: If an element $e := \langle k, p \rangle$ is redistributed into level i, exactly $2^i = \sum_{j<i} |D_j|$ elements $\langle \cdot, p' \rangle$ with $p' < p$ must have been redistributed into lower levels, so $2^i \leq \operatorname{rank}(e)$. Thus, for all non-dummy elements e inserted into a level $i \geq 1$, it holds that $\Delta_i \leq 2^i \leq \operatorname{rank}(e)$. For elements that remain in a level $i > m$, invariant (c) is trivially maintained. ◀

With this, we obtain our perfectly-secure priority queue construction:

▶ **Theorem 5** (Optimal Oblivious Priority Queue). *There is a deterministic, perfectly-secure oblivious priority queue with capacity N that supports each operation in amortized $\mathcal{O}(\log N)$ time and uses $\mathcal{O}(N)$ space.*

Proof. Apart from the rebuilding, the runtime for Insert, Min, and DeleteMin is constant. The amortized runtime per operation for Rebuild is bounded by

$$\sum_{m=0}^{\ell-1} \frac{T_{\text{KSelect}}(2^{m+1} + 2^m) + \sum_{i=0}^{m-1} T_{\text{KSelect}}(2^{i+2}) + c \cdot 2^m}{2^m}$$

$$\leq \sum_{m=0}^{\ell-1} \frac{\mathcal{O}(2^m)}{2^m} \in \mathcal{O}(\ell) = \mathcal{O}(\log N)\,.$$

The space bound follows immediately since all algorithmic building blocks have a linear runtime and the combined size of all up- and down-buffers is linear in N.

By using the deterministic, perfectly-secure algorithm for k-selection (Corollary 3), the obliviousness follows since the access pattern for each operation is a deterministic function of the capacity N and the number of operations performed so far. ◀

Due to the lower bound for oblivious priority queues [21], this runtime is optimal. If the type of operation does not need to be hidden, Min can be performed in constant time since rebuilding is only required for correctness when adding or removing an element. By applying Rebuild$(\ell - 1)$ directly, the priority queue can be initialized from up to N elements in $\mathcal{O}(N)$ time.

2.2 Non-Distinct Priorities

For non-distinct priorities, we want to ensure that ties are broken such that the order of insertion is preserved, i. e., that elements inserted earlier are extracted first. For this, we augment each element with the time-stamp t of the Insert operation and order the elements lexicographically by priority and time-stamp. This only increases the size of each element by a constant number of memory cells and thus does not affect the runtime complexity.

It remains to bound the size of the time-stamp for a super-polynomial number of operations:[6] Here we note that when rebuilding the last level (Rebuild$(\ell - 1)$), we can additionally sort all elements by time-stamp and compress the time-stamps to the range $\{0, \ldots, N-1\}$ (preserving their order). We then assign time-stamps starting with $t = N$ until

[6] Shi [29, Section III.E] also address this issue, but for our amortized construction we can use a simpler approach based on oblivious sorting.

Algorithm 2 Algorithm to perform an operation $\textsc{Op} \in \{\textsc{Read}, \textsc{Write}\}$ in the offline ORAM at the access location i; v' is the value to be written ($v' = \perp$ for $\textsc{Op} = \textsc{Read}$). The time-stamp t is incremented after each access (with $t = 1$ initially).

```
1: procedure $\textsc{Access}(\textsc{Op}, i, v')$
2:     $\langle v, t_{\text{next}} \rangle \leftarrow Q.\textsc{Min}()$
3:     $Q.\textsc{DeleteMin}()$ iff $t_{\text{next}} = t$; perform a dummy operation iff $t_{\text{next}} \neq t$
4:     $v \leftarrow \begin{cases} v & \text{iff } \textsc{Op} = \textsc{Read} \wedge t_{\text{next}} = t, \\ v_{\text{default}} & \text{iff } \textsc{Op} = \textsc{Read} \wedge t_{\text{next}} \neq t, \\ v' & \text{iff } \textsc{Op} = \textsc{Write} \end{cases}$
5:     $Q.\textsc{Insert}(v, \mathcal{T}[t-1])$; $t \leftarrow t + 1$        ▷ $\mathcal{T}[t-1] = \tau_t$
6:     return $v$
```

the last level is rebuilt again. This ensures that $2N$ is an upper bound for the time-stamps, so $\mathcal{O}(\log N)$ bits suffice for each time-stamp. With an (optimal) oblivious $\mathcal{O}(n \log n)$-time sorting algorithm [2], the additional amortized runtime for sorting is bounded by

$$\frac{1}{2^{\ell-1}} \cdot T_{\textsc{Sort}}(N) \in \Theta(\log N) \tag{2}$$

and does not affect the overall runtime complexity.

3 Offline ORAM from Oblivious Priority Queues

As mentioned in the introduction, an offline ORAM is an oblivious array data structure given the sequence $I = \langle i_1, \ldots, i_n \rangle$ of access locations in advance. While an offline ORAM may pre-process I to perform the operations more efficiently afterward, the data structure must still adhere to Definition 1, i. e., the memory probes must be independent of the values in I.

An offline ORAM can be constructed from any oblivious priority queue using a technique similar to *time-forward processing* [12]. We describe our construction for N memory cells below: In Section 3.1 we show how to realize $\textsc{Read}(i_t)$ and $\textsc{Write}(i_t, v'_t)$; there we assume that the time τ_t at which the index i_t is accessed next is known. In Section 3.2 we then show how to pre-process I to derive these values τ_t.

Jafargholi et al. [22] describe an alternative offline ORAM construction. We simplify the construction by decoupling the information τ_t from the values written to the offline ORAM.

3.1 Online Phase: Processing the Operations

For the offline ORAM with N cells, initialize a priority queue Q with capacity N; the annotations $\mathcal{T} = \langle \tau_1, \ldots, \tau_n \rangle$ as well as the current time t are stored alongside the priority queue. The procedure for processing the t-th operation $\textsc{Read}(i_t)$ or $\textsc{Write}(i_t, v'_t)$ is shown in Algorithm 2. Some fixed value v_{default} is used as the initial value of all ORAM cells.

It is easy to verify that the resulting construction is correct given $\mathcal{T}$ and oblivious given a perfectly-secure priority queue Q. For the array access $\mathcal{T}[t-1]$ in Line 5, note that t is the number of the current operation, so the access can be performed "in the clear" without a linear scan; this simplifies the construction w. r. t. Jafargholi et al. [22].

3.2 Offline Phase: Pre-Processing

To obtain the annotations $\mathcal{T} = \langle \tau_1, \ldots, \tau_n \rangle$ we now describe how to pre-process the sequence $I = \langle i_1, \ldots, i_n \rangle \in \{0, \ldots, N-1\}^n$ of memory access locations.

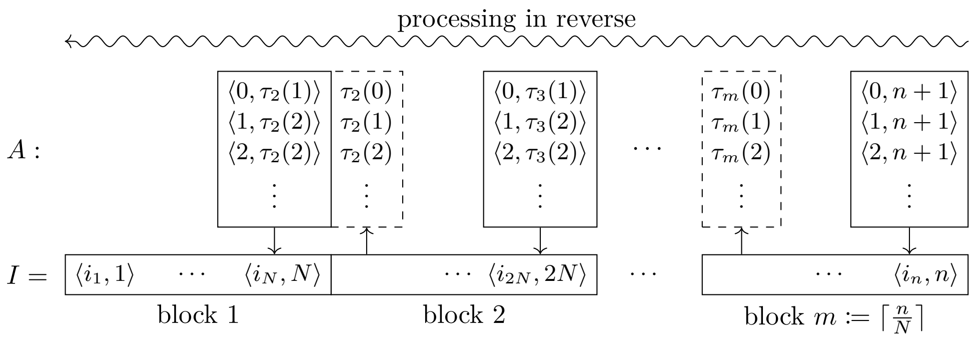

processing in reverse
A :
$\langle 0, \tau_2(1)\rangle$
$\langle 1, \tau_2(2)\rangle$
$\langle 2, \tau_2(2)\rangle$
$\tau_2(0)$
$\tau_2(1)$
$\tau_2(2)$
$\langle 0, \tau_3(1)\rangle$
$\langle 1, \tau_3(2)\rangle$
$\langle 2, \tau_3(2)\rangle$
$\tau_m(0)$
$\tau_m(1)$
$\tau_m(2)$
$\langle 0, n+1\rangle$
$\langle 1, n+1\rangle$
$\langle 2, n+1\rangle$
$I =$
$\langle i_1, 1\rangle$ $\cdots$ $\langle i_N, N\rangle$
$\cdots$ $\langle i_{2N}, 2N\rangle$
$\cdots$ $\langle i_n, n\rangle$
block 1
block 2
block $m := \lceil \frac{n}{N} \rceil$

Figure 3 Pre-processing the sequence of memory access locations I when n is super-polynomial in N. In this figure, $\langle i, t\rangle$ denotes a tuple of index i and time-stamp t while $\tau_j(i)$ denotes the time-stamp at which the index i is accessed next in block j.

The basic pre-processing proceeds as follows:

1. Annotate each index i_t with the time-stamp t.
2. Obliviously sort the indices I lexicographically by i_t and t.
3. Scan over indices in reverse, keeping track of the index i and the time-stamp t, and annotate each index i_t with the time-stamp τ_t it is accessed next (or some value larger than n if there is no next access).
4. Obliviously sort the indices I by t and discard everything but the annotations τ.

This can be done in amortized $\mathcal{O}(\log n)$ time per index with an $\mathcal{O}(n \log n)$-time oblivious sorting algorithm [2] and results in the annotations $\mathcal{T} = \langle \tau_1, \ldots, \tau_n\rangle$.

However, when the number of operations n is super-polynomial in the capacity N, i. e., when $n \in \omega(N^c)$ for all constants c, the time per index exceeds the optimal runtime of $\mathcal{O}(\log N)$. In this case, the pre-processing needs to be performed more carefully as shown in Figure 3 to maintain the amortized runtime of $\mathcal{O}(\log N)$: We divide the sequence I into blocks of size N. Additionally, we maintain an auxiliary block A with – for each index $i \in \{1, \ldots, N\}$ – the time-stamp τ at which index i is accessed next. Initially (for the last block), we initialize the time-stamp τ for each index i to some value greater than n.

The time-stamps τ_t are then determined block by block, from the last to the first. When processing each block, we update the time-stamps τ in A for the block processed next. For this, we can process the $\mathcal{O}(N)$ elements of each block (and A) as described above.

Since we process $\mathcal{O}(N)$ elements for each of the $\mathcal{O}(\lceil \frac{n}{N} \rceil)$ blocks by sorting and scanning, the pre-processing has a runtime of $\mathcal{O}(n \log N)$ overall. This maintains the desired runtime of $\mathcal{O}(\log N)$ per operation amortized.

With our priority queue construction from Section 2, we obtain the following:

▶ **Theorem 6** (Optimal Offline ORAM). *There is a deterministic, perfectly-secure offline ORAM with capacity N that has amortized $\mathcal{O}(\log N)$ overhead and uses $\mathcal{O}(N + n)$ space.*

Note that in contrast to the optimal statistically-secure constructions [22, 29], our offline ORAM maintains security and correctness for operation sequences of arbitrary length, e. g., when n is super-polynomial in N.

4 External-Memory Oblivious Priority Queue

In many applications of oblivious algorithms and data structures, e. g., for outsourced storage and for trusted computing in the presence of cache hierarchies, access to the main memory incurs high latencies. In these applications, the complexity of an algorithm is more

Table 3 Best known I/O upper bounds for obliviously partitioning n elements.

	cache-aware		*cache-agnostic*	
statistical security	$\mathcal{O}(\lceil \frac{n}{B} \rceil)$[t, w]	[17]	$\mathcal{O}(\lceil \frac{n}{B} \rceil)$[t, w]	[25]
perfect security	$\mathcal{O}(\lceil \frac{n}{B} \rceil \log_M n)$	[25]	$\mathcal{O}(\lceil \frac{n}{B} \rceil \log_M n)$	[25]
	$\mathcal{O}(\lceil \frac{n}{B} \rceil)$	**new, via** [4]	$\mathcal{O}(\lceil \frac{n}{B} \rceil \log \log_M n)$[t]	**new, via** [4]

[t] assuming a *tall cache* ($M \geq B^{1+\varepsilon}$) [w] assuming a *wide cache-line* ($B \geq \log^{\varepsilon} n$)

appropriately captured by the number of cache misses. This motivates the study of oblivious algorithms in the *external-memory* [1] and *cache-oblivious* [14] models; in this section we refer to these as *cache-aware* and *cache-agnostic* algorithms.

In this section, we instantiate I/O-efficient variants of our priority queue construction with perfect security and a private memory of constant size. For this, we sketch how to obtain I/O-efficient partitioning algorithms with perfect security in Section 4.1. We then analyze the I/O-efficiency of our priority queue construction in Section 4.2.

External-Memory Oblivious Algorithms. In cache-aware and cache-agnostic models, the CPU operates on the data stored in an *internal memory* (*cache*) of M memory words. Blocks (*cache-lines*) of B memory words can be transferred between the internal and a large external memory (*I/O operations*). The number of these I/O operations, depending on the problem size n as well as M and B, is the primary performance metric for external algorithms [1].

Cache-aware algorithms depend on the parameters M and B and explicitly issue the I/O operations. In contrast, cache-agnostic algorithms are unaware of the parameters M and B; here the internal memory is managed "automatically" through a *replacement policy* [14]. We assume an optimal replacement policy and a *tall cache*, i. e., $M \geq B^{1+\varepsilon}$ for a constant $\varepsilon > 0$; both are standard assumptions [14, 9].

For *oblivious external-memory* algorithms, we apply the Definition 1 to cache-aware and cache-agnostic algorithms. In both cases we assume that the internal memory is conceptually distinct from the constant-size private memory. That is, we guarantee that memory words both in the internal memory and within a block are accessed in an oblivious manner (*strong obliviousness* [9]). For this reason, our security definition remains unchanged. Note that this implies that the block access pattern is also oblivious, i. e., independent of the operations/inputs as in Definition 1.

Previous Work. While there is a line of research explicitly considering cache-aware [17, 18] and cache-agnostic [9, 25] oblivious algorithms, most works on oblivious algorithms consider internal algorithms with runtime and bandwidth overhead as performance metrics. To the best of our knowledge, cache-agnostic oblivious priority queues have not been explicitly considered in the literature. Implicitly, I/O-efficiency is sometimes [21, 22] treated through parameters: An oblivious algorithm with $(B \cdot w)$-width memory words and $M \cdot w$ bits of private memory can equivalently be stated as an oblivious external-memory algorithm with M words of internal memory and blocks of size B. We note that this re-parameterization does not allow distinguishing internal and private memory and that the resulting algorithms are inherently cache-aware.

Algorithm 3 Cache-aware oblivious partitioning algorithm. For simplicity, we assume $m \geq 2$.

```
procedure CacheAwarePartition$_P(A)$
    conceptually partition $A$ into $m := \lceil \frac{|A|}{B} \rceil$ blocks $G_i$ of $B$ consecutive elements each
    (where the last block may have fewer elements)
    for $i \leftarrow 0, \dots, m-1$ do Partition$_P(G_i)$
    for $i \leftarrow 1, \dots, m-2$ do PurifyHalf$_P(G_{i-1}, G_i)$          ▷ consolidate the blocks
    Partition$_{P' \colon X \mapsto P(X[0])}(\langle G_0, \dots, G_{m-3} \rangle)$          ▷ apply Partition to the blocks
    Reverse$(G_{m-1})$; PartBitonic$_P(G_{m-2} \parallel G_{m-1})$
    Reverse$(G_{m-2} \parallel G_{m-1})$; PartBitonic$_P(G_0 \parallel \cdots \parallel G_{m-1})$
```

This equivalence allows us to restate upper and lower bounds in terms of external-memory algorithms: Jacob et al. [21] show that $\Omega(\frac{1}{B} \log \frac{N}{M})$ I/O operations amortized are necessary for a cache-aware oblivious priority queue; this also applies to cache-agnostic oblivious priority queues.[7] This bound is matched by Jafargholi et al. [22], but the construction is cache-aware, requires $\Omega(\log N)$ words of private memory, and is randomized with statistical security.

The optimal internal partitioning algorithm [4] has – due to the use of expander graphs – an oblivious, but highly irregular access pattern and is thus not I/O-efficient. There are external oblivious partitioning algorithms [17, 25], but they are either only statistically secure or inefficient. We provide an overview of existing partitioning algorithms in Table 3.

4.1 External-Memory Oblivious Partitioning

For I/O-efficient instantiations of our priority queue, we need I/O-efficient partitioning algorithms. For this reason, we show how to construct an optimal cache-aware and a near-optimal cache-agnostic oblivious partitioning algorithm, respectively, with perfect security. Remember that for partitioning with a predicate P, we need to permute the elements such that all elements x with $P(x) = 0$ precede those with $P(x) = 1$.

We mainly rely on the optimal (internal) oblivious partitioning algorithm [4, Theorem 5.1] (Partition) and standard external-memory techniques. We also use oblivious building blocks from the previous work by Lin et al. [25, full version, Appendix C.1.2]:

PurifyHalf$_P(A, B)$. This procedure is given two partitioned blocks A and B with $|A| = |B|$ and permutes the elements such that A is *pure*, i. e., either only consists of elements x with $P(x) = 0$ or only consists of elements with $P(x) = 1$, and B is again partitioned.

PartBitonic$_P(A)$. This procedure is given a *bitonically partitioned* [25] array A, i. e., an array where all elements x with $P(x) = 1$ or all elements with $P(x) = 0$ are consecutive, and partitions A.

Both building blocks are deterministic with perfect security, cache-agnostic, and have a linear runtime of $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ [25].

Our cache-aware partitioning algorithm is shown in Algorithm 3. The idea is to split A into blocks of size B, partition each block, and then apply the internal partitioning algorithm to the blocks.

[7] In contrast, $\Theta(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/O operations amortized are sufficient for non-oblivious priority queues [3]. Here, the base of the logarithm is not constant but depends on the parameters M, B.

Algorithm 4 Cache-agnostic oblivious partitioning algorithm for $M \geq B^{1+\varepsilon}$. We assume $m \geq 2$.

```
1: procedure CacheAgnosticPartition_$P$($A$)
2:     if $n \leq 4$ then
3:         compact $A$ via oblivious sorting
4:     else
5:         conceptually partition $A$ into $m := \lceil \frac{|A|}{k} \rceil$ groups $G_i$ of $k := \lceil \sqrt[1+\varepsilon]{|A|} \rceil$ consecutive
           elements (where the last group may have fewer elements)
6:         for $i \leftarrow 0, \ldots, m-1$ do CacheAgnosticPartition_$P$($G_i$)
7:         for $i \leftarrow 1, \ldots, m-2$ do PurifyHalf_$P$($G_{i-1}, G_i$)      ▷ consolidate the groups
8:         Partition_$P' \colon X \mapsto P(X[0])$($\langle G_0, \ldots, G_{m-3} \rangle$)      ▷ apply Partition to the groups
9:         Reverse($G_{m-1}$); PartBitonic_$P$($G_{m-2} \parallel G_{m-1}$)
10:        Reverse($G_{m-2} \parallel G_{m-1}$); PartBitonic_$P$($G_0 \parallel \cdots \parallel G_{m-1}$)
```

▶ **Corollary 7** (Optimal Cache-Aware Oblivious Partitioning via [4, 25]). *There is a cache-aware, deterministic, perfectly-secure oblivious partitioning algorithm that requires $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ I/O operations for n elements.*

Proof. For the correctness, note that after Line 4 all blocks except G_{m-2} and G_{m-1} are pure. By applying the internal partitioning algorithm to the blocks in Line 5, all 0-blocks are swapped to the front. The partitioning is completed by first merging the partitions G_{m-2} and G_{m-1} in Line 6 and then merging both with the rest of the blocks in A.

The partitioning of each individual block is performed in internal memory and thus requires $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ I/O operations overall. The consolidation and merging of the partitions can also be performed with $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ I/O operations [25]. For the partitioning in Line 5, the I/O-efficiency follows from the construction of the internal partitioning algorithm [4, Theorem 5.1]: The algorithm operates in a "balls-in-bins"-manner, i. e., the elements are treated as indivisible. The algorithm performs a linear number of operations on $\lfloor \frac{n}{B} \rfloor$ elements, where each element has size $\mathcal{O}(B)$. This leads to an I/O complexity of $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ overall.

The obliviousness follows since the access pattern for each operation is a deterministic function of the input size $n := |A|$. ◀

For the cache-agnostic partitioning algorithm, the elements can be processed similarly. Here the parameter B is unknown, so the idea is to recursively divide into smaller groups until a group has size $\leq B$. The resulting algorithm is shown in Algorithm 4.

▶ **Corollary 8** (Cache-Agnostic Oblivious Partitioning via [4, 25]). *Assuming a tall cache of size $M \geq B^{1+\varepsilon}$ for a constant $\varepsilon > 0$, there is a cache-agnostic, deterministic, perfectly-secure oblivious partitioning algorithm that requires $\mathcal{O}(\lceil \frac{n}{B} \rceil \log \log_M n)$ I/O operations for n elements.*

Proof. The correctness of the base case is obvious. For the recursive case, the algorithm proceeds as the cache-aware Algorithm 3 above, so the correctness can be seen as in Corollary 7.

For the I/O complexity of Line 8, we distinguish two cases:

$k \leq B$. In this case $B \geq k \geq \sqrt[1+\varepsilon]{n}$, so $n \leq B^{1+\varepsilon} \leq M$ with the tall cache assumption. This means that the problem instance fits in the internal memory, and the step thus has an I/O complexity of $\mathcal{O}(\lceil \frac{n}{B} \rceil)$.

$k > B$. Here we can rely on the same insight as for the cache-aware partitioning, i. e., that the internal algorithm performs $\mathcal{O}(\frac{n}{k})$ operations on elements of size $k \geq B$. This leads to an I/O complexity of $\mathcal{O}(\frac{n}{k} \cdot \lceil \frac{k}{B} \rceil) = \mathcal{O}(\frac{n}{B})$.

The other steps have an I/O complexity of $\mathcal{O}(\lceil \frac{n}{B} \rceil)$ as in the cache-aware algorithm.

On depth i of the recursion tree, the instance size is

$$n_i := \sqrt[(1+\varepsilon)^i]{n} \qquad \text{so that on depth} \qquad i \geq \log_{1+\varepsilon} \frac{\log n}{\log M} \in \Theta(\log \log_M n)$$

the instances fit in the internal memory and no further I/O operations are required for the recursion. This leads to an I/O complexity of $\mathcal{O}(\lceil \frac{n}{B} \rceil \log \log_M n)$ overall.

As in Corollary 7, the obliviousness follows since the access pattern for each operation is a deterministic function of the input size $n := |A|$. ◀

4.2 Analysis of the External-Memory Oblivious Priority Queue

As for the internal algorithm introduced in Section 2, we can obtain efficient external oblivious algorithms for k-selection via partitioning [25, full version, Appendix E.2]. We thus obtain cache-aware and cache-agnostic algorithms for k-selection with the same asymptotic complexities as the partitioning algorithms described above.

With these external algorithms, we can analyze the construction described in Section 2 in the cache-aware and cache-agnostic settings:

▶ **Theorem 9** (External-Memory Oblivious Priority Queues). *There are deterministic, perfectly-secure oblivious priority queues with capacity N that support each operation with I/O complexity $\mathcal{O}(\frac{1}{B} \log \frac{N}{M})$ amortized (cache-aware) or $\mathcal{O}(\frac{1}{B} \log \frac{N}{M} \log \log_M N)$ amortized (cache-agnostic), respectively.*

Proof. We prove the theorem via a slightly more general statement: Assuming the existence of a deterministic, perfectly-secure k-selection algorithm with I/O complexity $O_{\textsc{KSelect}}(n) \in \Omega(\frac{n}{B})$, there is a deterministic, perfectly-secure priority queue with capacity N that supports each operation with I/O complexity $\mathcal{O}(\frac{O_{\textsc{KSelect}}(3N)}{N} \log \frac{N}{M})$ amortized.

Since the external k-selection algorithms are functionally equivalent to the internal algorithm and oblivious, the correctness and obliviousness follows from Theorem 5. For the I/O complexity, note that the first $j := \log_2 M - \mathcal{O}(1)$ levels of the priority queue fit into the M cells of the internal memory. The operations Insert, Min, and DeleteMin only operate on D_0 and U_0, so they do not require additional I/O operations.

It remains to analyze the I/O complexity of rebuilding the data structure. For this, we only need to consider rebuilding the levels $m \geq j$ since rebuilding levels $m < j$ only operates on the internal memory. When rebuilding a level $m \geq j$, the levels $i < j$ can be stored to and afterward retrieved from the external memory with $\mathcal{O}(\frac{M}{B})$ I/O operations. Assuming optimal page replacement, the amortized number of I/O operations for rebuilding is bounded by

$$\sum_{m=j}^{\ell-1} \frac{O_{\textsc{KSelect}}(2^{m+1}+2^m) + \sum_{i=0}^{m-1} O_{\textsc{KSelect}}(2^{i+2}) + c \cdot \frac{2^m}{B}}{2^m} + \underbrace{\mathcal{O}\left(\frac{M}{B \cdot 2^j}\right)}_{\text{levels } < j}$$

$$\leq \sum_{m=j}^{\ell-1} \frac{\mathcal{O}(O_{\textsc{KSelect}}(3 \cdot 2^m))}{2^m} + \mathcal{O}\left(\frac{1}{B}\right) \in \mathcal{O}\left((\ell - j) \cdot \frac{O_{\textsc{KSelect}}(3N)}{N}\right)$$

$$= \mathcal{O}\left(\frac{O_{\textsc{KSelect}}(3N)}{N} \log \frac{N}{M}\right).$$

With the cache-aware k-selection via Corollary 7 and the cache-agnostic k-selection via Corollary 8, we obtain the claimed I/O complexities. ◀

With this, we obtain an optimal cache-aware oblivious priority queue and a near-optimal cache-agnostic oblivious priority queue, both deterministic and with perfect security. Using a cache-agnostic, perfectly-secure oblivious sorting algorithm with (expected) I/O complexity $\mathcal{O}(\frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B})$ [9][8], we can apply the same construction as in Section 3 to obtain external offline ORAMs with the same (expected) I/O complexities as in Theorem 9. We exploit that the construction is a combination of sorting, linear scans, and time-forward processing.

5 Conclusion and Future Work

In this paper, we show how to construct an oblivious priority queue with perfect security and (amortized) logarithmic runtime. While the construction is simple, it improves the state-of-the-art for perfectly-secure priority queues, achieving the optimal runtime. The construction immediately implies an optimal offline ORAM with perfect security. We extend our construction to the external-memory model, obtaining optimal cache-aware and near-optimal cache-agnostic I/O complexities.

Future Work. The optimal perfectly-secure partitioning algorithm [4] has enormous constant runtime factors (in the order of $\gg 2^{111}$ [13]) due to the reliance on bipartite expander graphs.[9] Nevertheless, our construction can also be implemented efficiently in practice – albeit at the cost of an $\mathcal{O}(\log N)$-factor in runtime – by relying on merging [6] (instead of k-selection via linear-time partitioning). We leave comparing such a practical variant to previous protocols as a future work.

On the theoretical side, a main open problem is to obtain a perfectly-secure oblivious priority queue supporting deletions (of arbitrary elements) in optimal $\mathcal{O}(\log N)$ time. An additional open problem is the de-amortization of the runtime complexity. Considering external oblivious algorithms, an open problem is to close the gap on cache-oblivious partitioning, i. e., remove the remaining $\mathcal{O}(\log\log_M n)$ factor in I/O complexity for perfectly-secure algorithms. We consider all of these interesting problems for future works.

References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 Miklós Ajtai, Jánoss Komlós, and Endre Szemerédi. An $O(n\log n)$ sorting network. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM, 1983. doi:10.1145/800061.808726.

3 Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. Cache-oblivious priority queue and graph algorithm applications. In John H. Reif, editor, *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 268–276. ACM, 2002. doi:10.1145/509907.509950.

4 Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. *Journal of the ACM*, 70(1):4:1–4:70, 2022. doi:10.1145/3566049.

[8] Chan et al. [9] only describe a *statistically-secure* sorting algorithm that works by randomly permuting and then sorting the elements. Since the failure can only occur when permuting, can be detected, and leaks nothing about the input, we can repeat the permuting step until it succeeds [11, Section 3.2.2]. This leads to a perfectly-secure sorting algorithm with the same complexity in expectation.

[9] The algorithm of Dittmer and Ostrovsky [13] has lower constant runtime factors, but is randomized and only achieves statistical security due to a (negligible) failure probability.

5 Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, and Elaine Shi. Oblivious RAM with worst-case logarithmic overhead. *Journal of Cryptology*, 36(2):7, 2023. doi:10.1007/s00145-023-09447-5.

6 Kenneth E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. ACM, 1968. doi:10.1145/1468075.1468121.

7 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

8 Elette Boyle and Moni Naor. Is there an oblivious RAM lower bound? In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 357–368. ACM, 2016. doi:10.1145/2840728.2840761.

9 T.-H. Hubert Chan, Yue Guo, Wei-Kai Lin, and Elaine Shi. Cache-oblivious and data-oblivious sorting and applications. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2201–2220. SIAM, 2018. doi:10.1137/1.9781611975031.143.

10 T.-H. Hubert Chan and Elaine Shi. Circuit OPRAM: Unifying statistically and computationally secure ORAMs and OPRAMs. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, volume 10678 of *Lecture Notes in Computer Science*, pages 72–107. Springer, 2017. doi:10.1007/978-3-319-70503-3_3.

11 T.-H. Hubert Chan, Elaine Shi, Wei-Kai Lin, and Kartik Nayak. Perfectly oblivious (parallel) RAM revisited, and improved constructions. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*, volume 199 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITC.2021.8.

12 Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149. SIAM, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313681.

13 Samuel Dittmer and Rafail Ostrovsky. Oblivious tight compaction in $O(n)$ time with smaller constant. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 253–274. Springer, 2020. doi:10.1007/978-3-030-57990-6_13.

14 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297. IEEE, 1999. doi:10.1109/SFFCS.1999.814600.

15 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. doi:10.1080/00029890.1962.11989827.

16 Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996. doi:10.1145/233551.233553.

17 Michael T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In Rajmohan Rajaraman and Friedhelm Meyer auf der Heide, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 379–388. ACM, 2011. doi:10.1145/1989493.1989555.

18 Michael T. Goodrich and Joseph A. Simons. Data-oblivious graph algorithms in outsourced external memory. In Zhao Zhang, Lidong Wu, Wen Xu, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications*, volume 8881 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2014. doi:10.1007/978-3-319-12691-3_19.

19 Atsunori Ichikawa and Wakaha Ogata. Perfectly secure oblivious priority queue. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E106.A(3):272–280, 2023. doi:10.1587/transfun.2022CIP0019.

20 Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium*. The Internet Society, 2012. URL: https://www.ndss-symposium.org/ndss2012/access-pattern-disclosure-searchable-encryption-ramification-attack-and-mitigation.

21 Riko Jacob, Kasper Green Larsen, and Jesper Buus Nielsen. Lower bounds for oblivious data structures. In Timothy M. Chan, editor, *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2439–2447. SIAM, 2019. doi:10.1137/1.9781611975482.149.

22 Zahra Jafargholi, Kasper Green Larsen, and Mark Simkin. Optimal oblivious priority queues. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2366–2383. SIAM, 2021. doi:10.1137/1.9781611976465.141.

23 Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2014. doi:10.1007/978-3-662-45608-8_27.

24 Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology — CRYPTO 2018*, volume 10992 of *Lecture Notes in Computer Science*, pages 523–542. Springer, 2018. doi:10.1007/978-3-319-96881-0_18.

25 Wei-Kai Lin, Elaine Shi, and Tiancheng Xie. Can we overcome the $n \log n$ barrier for oblivious sorting? In Timothy M. Chan, editor, *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2419–2438. SIAM, 2019. doi:10.1137/1.9781611975482.148.

26 John C. Mitchell and Joe Zimmerman. Data-oblivious data structures. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *LIPIcs*, pages 554–565. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.554.

27 Arup Mondal, Priyam Panda, Shivam Agarwal, Abdelrahaman Aly, and Debayan Gupta. Fast and secure oblivious stable matching over arithmetic circuits. Cryptology ePrint Archive, Paper 2023/1789, 2023. URL: https://eprint.iacr.org/2023/1789.

28 Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *Topics in Cryptology — CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006. doi:10.1007/11605805_1.

29 Elaine Shi. Path oblivious heap: Optimal and practical oblivious priority queue. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 842–858. IEEE, 2020. doi:10.1109/SP40000.2020.00037.

30 Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 299–310. ACM, 2013. doi:10.1145/2508859.2516660.

31 Thore Thießen and Jan Vahrenhold. Optimal offline ORAM with perfect security via simple oblivious priority queues, 2024. doi:10.48550/arXiv.2409.12021.

32 Tomas Toft. Secure datastructures based on multiparty computation. Cryptology ePrint Archive, Paper 2011/081, 2011. URL: https://eprint.iacr.org/2011/081.

33 Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226. ACM, 2014. doi:10.1145/2660267.2660314.

Data Structures for Approximate Fréchet Distance for Realistic Curves

Ivor van der Hoog
DTU Compute, Technical University of Denmark, Lyngby, Denmark

Eva Rotenberg
DTU Compute, Technical University of Denmark, Lyngby, Denmark

Sampson Wong
Department of Computer Science, University of Copenhagen, Denmark

Abstract

The Fréchet distance is a popular distance measure between curves P and Q. Conditional lower bounds prohibit $(1+\varepsilon)$-approximate Fréchet distance computations in strongly subquadratic time, even when preprocessing P using any polynomial amount of time and space. As a consequence, the Fréchet distance has been studied under *realistic* input assumptions, for example, assuming both curves are *c-packed*.

In this paper, we study c-packed curves in Euclidean space $\mathbb{R}^d$ and in general geodesic metrics $\mathcal{X}$. In $\mathbb{R}^d$, we provide a nearly-linear time static algorithm for computing the $(1+\varepsilon)$-approximate continuous Fréchet distance between c-packed curves. Our algorithm has a linear dependence on the dimension d, as opposed to previous algorithms which have an exponential dependence on d.

In general geodesic metric spaces $\mathcal{X}$, little was previously known. We provide the first data structure, and thereby the first algorithm, under this model. Given a c-packed input curve P with n vertices, we preprocess it in $O(n \log n)$ time, so that given a query containing a constant ε and a curve Q with m vertices, we can return a $(1+\varepsilon)$-approximation of the discrete Fréchet distance between P and Q in time polylogarithmic in n and linear in m, $1/\varepsilon$, and the realism parameter c.

Finally, we show several extensions to our data structure; to support dynamic extend/truncate updates on P, to answer map matching queries, and to answer Hausdorff distance queries.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, data structures, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.56

Related Version *Full Version*: https://arxiv.org/abs/2407.05114 [44]

Funding *Ivor van der Hoog*: Supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis".
Eva Rotenberg: Partially supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis" and the Carlsberg Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications".

1 Introduction

The Fréchet distance is a popular metric for measuring the similarity between (polygonal) curves P and Q. We assume that P has n vertices and Q has m vertices and that they reside in some geodesic metric space $\mathcal{X}$. The Fréchet distance is often intuitively defined through the following metaphor: suppose that we have two curves that are traversed by a person and their dog. Consider the length of their connecting leash, measured over the metric $\mathcal{X}$. What is the minimum length of the connecting leash over all possible traversals by the person and the dog? The Fréchet distance has many applications; in particular in the analysis and

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 56; pp. 56:1–56:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

visualization of movement data [11, 14, 35, 47]. It is a versatile measure that can be used for a variety of objects, such as handwriting [41], coastlines [37], outlines of shapes in geographic information systems [20], trajectories of moving objects, such as vehicles, animals or sports players [40, 42, 7, 14], air traffic [6] and protein structures [34].

Alt and Godau [2] compute the continuous Fréchet distance in $\mathbb{R}^2$ under the L_2 metric in $O(mn \log(n+m))$ time. This was later improved by Buchin et al. [12] to $O(nm(\log\log nm)^2)$ time. Eiter and Manila [26] showed how to compute the discrete Fréchet distance in $\mathbb{R}^2$ in $O(nm)$ time, which was later improved by Agarwal et al. [1] to $O(nm(\log\log nm)/\log nm)$ time. Typically, the quadratic $O(nm)$ running time is considered costly. Bringmann [8] showed that, conditioned on the Strong Exponential Time Hypothesis (SETH), one cannot compute a $(1+\varepsilon)$-approximation of the continuous Fréchet distance between curves in $\mathbb{R}^2$ under the L_1, L_2 or L_∞ metric faster than $\Omega((nm)^{1-\delta})$ time for any $\delta > 0$. This lower bound was extended by Buchin, Ophelders and Speckmann [13] to intersecting curves in $\mathbb{R}^1$. Driemel, van der Hoog and Rotenberg [25] extended the lower bound to paths P and Q in a weighted planar graph under the shortest path metric.

Well-behaved curves. Previous works have circumvented lower bounds by assuming that *both* curves come from a well-behaved class. A curve P in a geodesic metric space $\mathcal{X}$ is any sequence of points where consecutive points are connected by their shortest path in $\mathcal{X}$. For a ball B in $\mathcal{X}$, let $P \cap B$ denote all (maximal) segments of P contained in B. A curve P is:

- **κ-straight** (by Alt, Knauer and Wenk [3]) if for every i, j the length of the subcurve from p_i to p_j is $\ell(P[i,j]) \leq \kappa \cdot d(p_i, p_j)$,
- **c-packed** (by Driemel, Har-Peled and Wenk [22]) if for every ball B in the geodesic metric space $\mathcal{X}$ with radius r: the length $\ell(P \cap B) \leq c \cdot r$.
- **ϕ-low-dense** (by van der Stappen [45]; see also [20, 22, 39]) if for every ball B in $\mathcal{X}$ with radius r, there exist at most ϕ edges of length r intersecting B.
- **backbone** (by Aronov et al. [5]) if consecutive vertices have distance between c_1 and c_2 for some constants c_1, c_2, and if non-consecutive vertices have distance at least 1.

Any c-straight curve is also $O(c)$-packed. Parametrized by ε, $\phi \in O(1)$, c and $\kappa = O(c)$, Driemel, Har-Peled and Wenk [22] compute a $(1+\varepsilon)$-approximation of the continuous Fréchet distance between a pair of realistic curves in $\mathbb{R}^d$ under the L_1, L_2, L_∞ metric for constant d in $O(\frac{c(n+m)}{\varepsilon} + c(n+m)\log n)$ time. Their result for c-packed and c-straight curves was improved by Bringmann and Künnemann [10] to $O(\frac{c(n+m)}{\sqrt{\varepsilon}} \log \varepsilon^{-1} + c(n+m)\log n)$, which matches the conditional lower bound for c-packed curves. In particular, Bringmann [8] showed that under SETH, for dimension $d \geq 5$, there is no $O((c(n+m)/\sqrt{\varepsilon})^{1-\delta})$ time algorithm for computing the Fréchet distance between c-packed curves for any $\delta > 0$. Realistic input assumptions have been applied to other geometric problems, e.g. for robotic navigation in ϕ-low-dense environments [45], and map matching of ϕ-low-dense graphs [16] or c-packed graphs [30].

Deciding versus computing. We make a distinction between two problem variants: the decision variant, the optimisation variant. For the decision variant, we are given a value ρ and two curves P and Q and we ask whether the Fréchet distance $D_{\mathcal{F}}(P,Q) \leq \rho$. This variant often solved through navigating an n by m "free space diagram". In the optimization variant, the goal is to output the Fréchet distance $D_{\mathcal{F}}(P,Q)$. To convert any decision algorithm into a optimization algorithm, two techniques are commonly used. The first is binary search over what we will call $\text{TADD}(P,Q)$:

▶ **Definition 1.** *Given two sets of points P, Q in a geodesic metric space $\mathcal{X}$, we define a Two-Approximate Distance Decomposition of P denoted by* ***TADD(P, Q)*** *as a set of reals T_{PQ} where for every pair $(p_i, q_j) \in P \times Q$ there exist $a, b \in T_{PQ}$ with $a \leq d(p_i, q_j) \leq b \leq 2a$.*

Essentially a TADD is a two-approximation of the set of all pairwise distances in $P \times Q$ and it can be used to determine, approximately, the (Fréchet) distance values for when the simplification of the input curve changes, or when the reachability of the free space matrix changes. It is known how to compute a TADD from a Well-Separated Pair Decomposition (WSPD) in time linear in the size of the WSPD [22, Lemma 3.8]. A downside of this approach [22] is that, it is only known how to compute a WSPD for doubling metrics [48]. Moreover, for non-constant (doubling) dimensions d, computing the WSPD (and therefore the TADD) takes $O(2^d n + dn \log n)$ time [33, 48], which dominates the running time.

The second technique, deployed when for example TADDs cannot be computed, is parametric search [38]. For decision variants that have a sublinear running or query time of T, the running time of parametric search is commonly $O(T^2)$ [46, 31].

Data structures for Fréchet distance. An interesting question is whether we can store P in a data structure, for efficient (approximate) Fréchet distance queries for any query Q. This topic received considerable attention throughout the years [24, 31, 27, 19, 21, 15, 30]. A related field is nearest neighbor data structures under the Fréchet distance metric [9, 23, 18, 4, 28]. Recently, Gudmundsson, Seybold and Wong [30] answer this question negatively for arbitrary curves in $\mathbb{R}^2$: showing that even with polynomial preprocessing space and time, we cannot preprocess a curve P to decide the continuous Fréchet distance between P and a query curve Q in $\Omega((nm)^{1-\delta})$ time for any $\delta > 0$. Surprisingly, even in very restricted settings data structure results are difficult to obtain. De Berg et al. [19] present an $O(n^2)$ size data structure that restricts the orientation of the query segment to be horizontal. Queries are supported in $O(\log^2 n)$ time, and even subcurve queries are allowed (in that case, using $O(n^2 \log^2 n)$ space). At ESA 2022, Buchin et al. [15] improve these result to using only $O(n \log^2 n)$ space, where queries take $O(\log n)$ time. For arbitrary query segments, they present an $O(n^{4+\delta})$ size data structure that supports (subcurve) queries to arbitrary segments in $O(\log^4 n)$ time. Gudmundsson et al. [31] extend de Berg et al.'s [19] data structure to handle subcurve queries, and to handle queries where the horizontal query segment is translated in order to minimize its Fréchet distance. Driemel and Har-Peled [21] create a data structure to store any curve in $\mathbb{R}^d$ for constant d. They preprocess P in $O(n \log^3 n)$ time and $O(n \log n)$ space. For any query (Q, ε, i, j) they can create a $(3 + \varepsilon)$-approximation of $D_{\mathbb{F}}(P[i, j], Q)$ in $O(m^2 \log n \log(m \log n))$ time.

We state existing data structures for the discrete Fréchet distance. Driemel, Psarros and Schmidt [24] fix ε and an upper bound M beforehand where for all queries Q, they demand that $|Q| \leq M$. They store any curve P in $\mathbb{R}^d$ for constant d using $O((M \log \frac{1}{\varepsilon})^M)$ space and preprocessing, to answer $(1 + \varepsilon)$-approximate Fréchet distance queries in $O(m^2 + \log \frac{1}{\varepsilon})$ time. Filtser [29] gives the corresponding data structure for the discrete Fréchet distance. At SODA 2022, Filtser and Filtser [27] study the same setting: storing P in $O((\frac{1}{\varepsilon})^{dM} \log \frac{1}{\varepsilon})$ space, to answer $(1 + \varepsilon)$-approximate Fréchet distance queries in $\tilde{O}(m \cdot d)$ time.

Contributions. We provide four contributions.

(1) A 1-TADD technique. A crucial step in computing the Fréchet distance is to turn a decision algorithm into an optimization algorithm. TADD(P, Q) is commonly used when approximating the Fréchet distance. Our 1-TADD technique shows a new argument where we map P and Q to curves in $\Lambda \subset \mathbb{R}^1$ and compute only TADD(Λ, Λ) in $O(n \log n)$ time.

Table 1 Our results for computing a $(1+\varepsilon)$-approximation of the Fréchet distance between P and Q. All settings assume that P is a realistic curve, except for [27] who assume an upper bound on $|Q|$. T_ε denotes the query time of a $(1+\varepsilon)$-approximate oracle. The tilde hides lower order factors in terms of n, m and ε. Under the continuous Fréchet distance, we require that Q is also realistic.

Domain	Previous result			Our result		
	Preprocess	Query time	Ref.	Preprocess	Query time	Ref.
$\mathbb{R}^d$ L_p metric	static	$\tilde{O}(2^d n + d^{\frac{c(n+m)}{\sqrt{\varepsilon}}}$ $+d^2 \cdot c(n+m))$	[10]	static	$\tilde{O}(d^{\frac{c(n+m)}{\varepsilon}})$	Thm. 2
Discrete Fréchet distance only:						
$\mathbb{R}^d$ $\lvert Q\rvert \leq M$	$\tilde{O}((\frac{1}{\varepsilon})^{dM})$	$\tilde{O}(md)$	[27]	$O(n \log n)$	$\tilde{O}(d\frac{cm}{\varepsilon}\log n)$	Cor. 7
Planar $G=(V,E)$	static	$\tilde{O}(\lvert V\rvert^{1+o(1)} + \frac{c(n+m)}{\varepsilon})$	[25]	$\tilde{O}(\lvert V\rvert^{1+o(1)})$	$\tilde{O}(\frac{cm}{\varepsilon}\log n)$	Cor. 7
Graph $G=(V,E)$	static	$\tilde{O}(\lvert V\rvert^{1+o(1)} + \lvert E\rvert \log \lvert E\rvert$ $+\frac{c(n+m)}{\varepsilon})$	[25]	$\tilde{O}(\lvert V\rvert^{1+o(1)})$	$\tilde{O}(\frac{cm}{\varepsilon}\log n)$	Cor. 7
Simple Polygon P	static	$O(nm\log(n+m))$	[2]	$\tilde{O}(\lvert P\rvert + n)$	$\tilde{O}(\frac{cm}{\varepsilon}\log \lvert P\rvert)$	Cor. 7
Any geodesic $\mathcal{X}$ with $(1+\varepsilon)$-oracle	static	$O(T_\varepsilon \cdot nm \log n)$	[26]	$O(n \log n)$	$\tilde{O}(T_\varepsilon \frac{cm}{\varepsilon}\log n)$	Thm. 5
Map Matching $G=(V,E)$	$\tilde{O}(c^2\varepsilon^{-4}n^2)$	$O(m\log m\cdot$ $(\log^4 n + c^4\varepsilon^{-8}\log^2 n))$	[30]	$\tilde{O}(c^2\varepsilon^{-4}n^2)$	$O(m(\log n + \log\varepsilon^{-1})\cdot$ $(\log^2 n + c^2\varepsilon^{-4}\log n))$	[44]

In Euclidean $\mathbb{R}^d$ this allows us to approximate the discrete and continuous Fréchet distances in time that is linear in d, whereas previous approaches required an exponential dependence on d. In general geodesic metric spaces $\mathcal{X}$, our 1-TADD technique allows us to approximate the discrete Fréchet distance when $\text{TADD}(P,Q)$ cannot be efficiently computed.

(2) Allowing approximate oracles under the discrete Fréchet distance. Many ambient spaces (e.g., Euclidean spaces under floating point arithmetic, and $\mathcal{X}$ as a weighted graph under the shortest path metric.) do not allow for efficient exact distance computations. Thus, we revisit and simplify the argument by Driemel, Har-Peled and Wenk [22]. We assume access to a $(1+\alpha)$-approximate distance oracle with T_α query time. We generalize the previous argument to approximate the discrete Fréchet distance between two curves in any geodesic metric space with approximate distance oracles. For contributions *(1)* and *(2)*, we do not require the curves P and Q to be c-packed.

(3) A data structure under the discrete Fréchet distance. Under the discrete Fréchet distance, we show how to store a c-packed or c-straight curve P with n vertices in *any* geodesic ambient space $\mathcal{X}$. Our solution uses $O(n)$ space and $O(n\log n)$ preprocessing time. For *any* query curve Q, any $0<\varepsilon<1$, and any subcurve P^* of P, we can compute a $(1+\varepsilon)$-approximation the discrete Fréchet distance $D_{\mathbb{F}}(P^*,Q)$ using $O(\frac{c\cdot m}{\varepsilon}\log n(T + \log\frac{c\cdot m}{\varepsilon} + \log n))$ time. Here, T is the time required to perform a distance query in the ambient space (e.g., $O(\log n)$ for geodesic distances in a polygon). All times are deterministic and worst-case. This is the Fréchet distance first data structure for realistic curves that avoids spending query time linear in n. Our solution improves various recent results [10, 25, 27, 30] (see Table 1).

(4) Extensions. In the full version of this paper [44], we provide several extensions to our results. We modify our data structure to support updates that truncate the curve P, or extend P, we apply our algorithmic skeleton to map matching queries, and we study Hausdorff distance queries.

2 Preliminaries

Let $\mathcal{X}$ denote some geodesic metric space (e.g., $\mathcal{X}$ is some weighted graph). For any $a, b \in \mathcal{X}$ we denote by $d(a, b)$ their distance in $\mathcal{X}$. A *curve* P in $\mathcal{X}$ is any ordered sequence of points in $\mathcal{X}$, where consecutive points are connected by their shortest path in $\mathcal{X}$. We refer to such points as *vertices*. For any curve P with n vertices, for any integers $i, j \in [n]$ with $i < j$ we denote by $P[i, j]$ the subcurve from p_i to p_j. We denote by $|P[i, j]| = (j - i + 1)$ the size of the subcurve and by $\ell(P[i, j]) := \sum_{k=i}^{j-1} d(p_k, p_{k+1})$ its length. We receive as preprocessing input a curve P where for each pair (p_i, p_{i+1}) we are given $d(p_i, p_{i+1})$.

Distance and distance oracles. Throughout this paper, we assume that for any $\alpha > 0$ we have access to some $(1 + \alpha)$-approximate distance oracle. This is a data structure $\mathcal{D}^{\alpha}_{\mathcal{X}}$ that for any two $a, b \in \mathcal{X}$ can report a value $d^{\circ}(a, b) \in [(1 - \alpha)d(a, b), (1 + \alpha)d(a, b)]$ in $O(T_\alpha)$ time. To distinguish between inaccuracy as a result of our algorithm and as a result of our oracle, we refer to $d^{\circ}(a, b)$ as the *perceived value* (as opposed to an approximate value).

Discrete Fréchet distance. Given two curves P and Q in $\mathcal{X}$, we denote by $[n] \times [m]$ the n by m integer lattice. We say that an ordered sequence F of points in $[n] \times [m]$ is a *discrete walk* if for every consecutive pair $(i, j), (k, l) \in F$, we have $k \in \{i - 1, i, i + 1\}$ and $l \in \{j - 1, j, j + 1\}$. It is furthermore *$xy$-monotone* when we restrict to $k \in \{i, i + 1\}$ and $l \in \{j, j + 1\}$. Let F be a discrete walk from $(1, 1)$ to (n, m). The *cost* of F is the maximum over $(i, j) \in F$ of $d(p_i, q_j)$. The discrete Fréchet distance is the minimum over all xy-monotone walks F from $(1, 1)$ to (n, m) of its associated cost: $D_{\mathbb{F}}(P, Q) := \min_F \text{cost}(F) = \min_F \max_{(i,j) \in F} d(p_i, q_j)$. In this paper we, given a $(1 + \alpha)$-approximate distance oracle, define the *perceived* discrete Fréchet distance as $D_{\mathbb{F}}{}^{\circ}$, obtained by replacing in the above definition $d(p_i, q_j)$ by $d^{\circ}(p_i, q_j)$.

Free space matrix (FSM). The FSM for a fixed $\rho^* \geq 0$ is a $|P| \times |Q|$, $(0, 1)$-matrix where the cell (i, j) is zero if and only if the distance between the i'th point in P and the j'th point in Q is at most ρ^*. Per definition, $D_{\mathbb{F}}(P, Q) \leq \rho^*$ if and only if there exists an xy-monotone discrete walk F from $(1, 1)$ to (n, m) where for all $(i, j) \in F$: the cell (i, j) is zero.

Continuous Fréchet distance and Free Space Diagram (FSD). We define the continuous Fréchet distance in a geodesic metric space. Given a curve P, we consider P as a continuous function mapping at time $t \in [0, 1]$ to a point $P(t)$ in $\mathcal{X}$. The continuous Fréchet distance is $D_{\mathcal{F}}(P, Q) := \inf_{\alpha,\beta} \max_{t \in [0,1]} d(P(\alpha(t)), Q(\beta(t)))$, where $\alpha, \beta : [0, 1] \to [0, 1]$ are non-decreasing surjections. For a fixed ρ^*, we can define the Free Space Diagram of (P, Q, ρ^*) to be a $[0, 1] \times [0, 1]$, (0,1)-matrix where the point (t, t') is zero if and only if the distance between $P(t)$ and $Q(t')$ is at most ρ^*. The diagram consists of nm cells corresponding to all pairs of edges of P and Q. A cell is *reachable* if there exists an xy-monotone curve from $(0, 0)$ to a point in the cell where all points (t, t') on the curve are zero. The continuous Fréchet distance is at most ρ^* if and only if there exists an xy-monotone curve from $(0, 0)$ to $(1, 1)$ where all points (t, t') on the curve are zero.

Defining discrete queries. Our data structure input is a curve $P = (p_1, p_2, \ldots, p_n)$. The number of vertices m of Q is part of the query input and may vary. Let $D_{\mathbb{F}}(P, Q)$ denote the discrete Fréchet distance between P and Q. We distinguish between four types of (approximate) queries. The input pararmeters are given at query time:

- **A-decision($\boldsymbol{Q, \varepsilon, \rho}$):** for $\rho \geq 0$ and $0 < \varepsilon < 1$ outputs a Boolean concluding either $D_{\mathbb{F}}(P, Q) > \rho$, or $D_{\mathbb{F}}(P, Q) \leq (1 + \varepsilon)\rho$ (these two options are not mutually exclusive).
- **A-value($\boldsymbol{Q, \varepsilon}$):** for $0 < \varepsilon < 1$ outputs a value in $[(1 - \varepsilon)D_{\mathbb{F}}(P, Q), (1 + \varepsilon)D_{\mathbb{F}}(P, Q)]$.
- **Subcurve-decision($\boldsymbol{Q, \varepsilon, \rho, i, j}$):** for $\rho \geq 0$ and $0 < \varepsilon < 1$ outputs a Boolean concluding either $D_{\mathbb{F}}(P[i, j], Q) > \rho$, or $D_{\mathbb{F}}(P[i, j], Q) \leq (1 + \varepsilon)\rho$.
- **Subcurve-value($\boldsymbol{Q}$, $\boldsymbol{\varepsilon}$, $\boldsymbol{i}$, $\boldsymbol{j}$):** for $0 < \varepsilon < 1$ outputs a value in $[(1 - \varepsilon)D_{\mathbb{F}}(P[i, j], Q), (1 + \varepsilon)D_{\mathbb{F}}(P[i, j], Q)]$.

We want a solution that is efficient in time and space, where time and space is measured in units of ε, n, m, ρ and the distance oracle query time T_α.

Previous works: $\boldsymbol{\mu}$-simplifications. Driemel, Har-Peled and Wenk [22] , for a parameter $\mu \in \mathbb{R}$, construct a curve P^μ as follows. Start with the initial vertex p_1, and set this as the current vertex p_i. Next, scan the polygonal curve to find the first vertex p_j such that $d(p_i, p_j) > \mu$. Add p_j to P^μ, and set p_j as the current vertex. Continue this process until we reach the end of the curve. Finally, add the last vertex p_n to P^μ. Driemel, Har-Peled and Wenk [22] observe any μ-simplified curve P^μ can be computed in linear time and $D_{\mathcal{F}}(P^\mu, P) \leq \mu$. This leads to the following approximate decision algorithm:

1. Given P, ε, Q and ρ, construct $P^{\frac{\varepsilon\rho}{4}}$ and $Q^{\frac{\varepsilon\rho}{4}}$ in $O(n + m)$ time.
2. Denote by X the reachable cells in the FSD of $(P^{\frac{\varepsilon\rho}{4}}, Q^{\frac{\varepsilon\rho}{4}}, \rho^* = (1 + \varepsilon/2)\rho)$.
3. Iterating over X, doing $O(|X|)$ distance computations, test if $D_{\mathcal{F}}(P^{\frac{\varepsilon\rho}{4}}, Q^{\frac{\varepsilon\rho}{4}}) \leq \rho^*$.
   - They prove that: if yes then $D_{\mathcal{F}}(P, Q) \leq (1 + \varepsilon)\rho$. If no then $D_{\mathcal{F}}(P, Q) > \rho$.
   - If P and Q are c-packed, they upper bound $|X|$ by $O(\frac{c(n+m)}{\varepsilon})$.

This scheme is broadly applicable to various domains, see [25, 17]. In this paper, we apply this technique to answer value queries at the cost of a factor $O(\log n + \log \varepsilon^{-1})$. Under the discrete Fréchet distance, we extend the analysis to work with approximate distance oracles. Finally, we show a data structure to execute step 3 in time independent of $|P| = n$. We also show that under the discrete Fréchet distance, it suffices to assume that only P is c-packed.

2.1 Results

(1) A 1-TADD technique for the Fréchet distance. For any $\mu > 0$, we denote by P^μ and Q^μ their μ-simplified curves according to our new definition of μ-simplification. We show in Section 4 that our new definition allows us to efficiently transform existing decision algorithms into approximation algorithms in $\mathbb{R}^d$ [22]. We assume access to exact $O(d)$-time distance oracle in $\mathbb{R}^d$ and prove:

▶ **Theorem 2.** *We can preprocess a pair of c-packed curves (P, Q) in $\mathbb{R}^d$ under any L_p metric with $|P| = n \geq |Q| = m$ in $O(n \log n)$ time s.t.: given any ε and an exact distance oracle, we can compute a $(1 + \varepsilon)$-approximation of $D_{\mathcal{F}}(P, Q)$ in $O(d\frac{c(n+m)}{\varepsilon} \cdot (\log n + \log \varepsilon^{-1}))$ time.*

(2) Allowing approximate oracles under the discrete Fréchet distance. In Section 5, we show that (for computing the discrete Fréchet distance) it suffices to have access to a $(1 + \alpha)$-approximate distance oracle. This will enable us to approximate $D_{\mathbb{F}}(P, Q)$ in ambient spaces such as planar graphs and simple polygons. Formally, we show:

▶ **Lemma 3.** *For any $\rho > 0$ and $0 < \varepsilon < 1$, choose $\rho^* = (1 + \frac{1}{2}\varepsilon)\rho$ and $\mu \leq \frac{1}{6}\varepsilon\rho$. Let $\mathcal{X}$ be any geodesic metric space and $\mathcal{D}_{\mathcal{X}}^{\varepsilon/6}$ be a $(1 + \frac{1}{6}\varepsilon)$-approximate distance oracle. For any curve $P = (p_1, \dots, p_n)$ in $\mathcal{X}$ and any curve $Q = (q_1, \dots, q_m)$ in $\mathcal{X}$:*

- *If for the discrete Fréchet distance, ${D_{\mathbb{F}}}^{\circ}(P^{\mu}, Q) \leq \rho^*$ then $D_{\mathbb{F}}(P, Q) \leq (1 + \varepsilon)\rho$.*
- *If for the discrete Fréchet distance, ${D_{\mathbb{F}}}^{\circ}(P^{\mu}, Q) > \rho^*$ then $D_{\mathbb{F}}(P, Q) > \rho$.*

We note that for ambient spaces such as planar graphs and simple polygons, there is no clear way to define a continuous μ-simplification (or even continuous Fréchet distance).

(3) An efficient data structure under the discrete Fréchet distance. Finally, in Section 6, we study computing the discrete Fréchet distance in a data structure setting. We show that under the discrete Fréchet distance, it suffices to assume that only P is c-packed or c-straight. Moreover, we can store P in a data structure such that we can answer approximate Fréchet value queries $D_{\mathbb{F}}(P, Q)$ in time linear in m and polylogarithmic in n:

▶ **Theorem 4.** *Let $\mathcal{X}$ be any geodesic space and $\mathcal{D}_{\mathcal{X}}^{\alpha}$ be a $(1+\alpha)$-approximate distance oracle with $O(T_\alpha)$ query time. Let $P = (p_1, \dots, p_n)$ be any c-packed curve in $\mathcal{X}$. We can store P using $O(n)$ space and preprocessing, such that for any curve $Q = (q_1, \dots, q_m)$ in $\mathcal{X}$ and any $\rho > 0$ and $0 < \varepsilon < 1$, we can answer A-decision(Q, ε, ρ) for the discrete Fréchet distance in:*

$$O\left(\frac{c \cdot m}{\varepsilon} \cdot (T_{\varepsilon/6} + \log n)\right) \text{ time.}$$

We may apply the proof of Theorem 2 to Theorem 4 to answer A-value(Q, ε) in any geodesic metric space by increasing the running time by a factor $O(\log n + \log \varepsilon^{-1})$. However, under the discrete Fréchet distance we can be more efficient:

▶ **Theorem 5.** *Let $\mathcal{X}$ be a geodesic metric space and $\mathcal{D}_{\mathcal{X}}^{\alpha}$ be a $(1 + \alpha)$-approximate distance with $O(T_\alpha)$ query time. Let $P = (p_1, \dots, p_n)$ be any c-packed curve in $\mathcal{X}$. We can store P using $O(n)$ space and $O(n \log n)$ preprocessing time, such that for any curve $Q = (q_1, \dots, q_m)$ in $\mathcal{X}$ and any $0 < \varepsilon < 1$, we can answer A-Value(Q, ε) for the discrete Fréchet distance in:*

$$O\left(\frac{c \cdot m}{\varepsilon} \cdot \log n \cdot \left(T_{\varepsilon/6} + \log \frac{c \cdot m}{\varepsilon} + \log n\right)\right) \text{ time.}$$

The advantage of our result is that it applies to a variety of metric spaces $\mathcal{X}$, while also improving upon previous *static* algorithms in those spaces. Here, static refers to solutions that do not require preprocessing or building a data structure. For a complete overview of the improvements we make to the state-of-the-art, we refer to Table 1.

Our result does not rely upon complicated techniques such parametric search [31, 30], higher-dimensional envelopes [15], or advanced path-simplification structures [21, 24, 27]. Our techniques are not only generally applicable, but also appear implementable (e.g., the authors of [15] mention that their result is un-implementable).

2.2 Corollaries

Theorems 4+5 are the first data structures for computing the Fréchet distance between c-packed curves. Our construction has two novelties: first, we consider c-packed curves in *any* metric space $\mathcal{X}$, and only require access to *perceived* distances (distance oracles that can report a $(1 + \alpha)$-approximation in $O(T_\alpha)$ time). Second, we propose a new 1-TADD technique that can be used to compute the discrete Fréchet distance independently of the geodesic ambient space $\mathcal{X}$. These two novelties imply improvements to previous results, even static results for computing the Fréchet distance:

Applying perceived distances. Our analysis allows us to answer the decision variant for the discrete Fréchet distance for any geodesic metric space for which there exist efficient $(1+\alpha)$-approximate distance oracles (See Oracles 12). Combining Theorem 4 with Oracles 12 we obtain:

▶ **Corollary 6.** *Let P be a c-packed curve in a metric space $\mathcal{X}$.*

- *For $\mathcal{X} = \mathbb{R}^d$ under L_1, L_2, L_∞ metric in the real-RAM model, we can store P using $O(n)$ space and preprocessing, to answer A-decision(Q, ε, ρ) in $O\left(\frac{cm}{\varepsilon} \cdot (d + \log n)\right)$ time.*
  - *Improving the static $O(d \cdot \frac{cn}{\sqrt{\varepsilon}} + d \cdot cn \log n)$ algorithm by Bringmann and Künnemann [10, IJCGA'17]: making it faster when $n > m$ and making it a data structure.*
- *For $\mathcal{X} = \mathbb{R}^d$ under L_2 in word-RAM, we can store P using $O(n)$ space and $O(n \log n)$ preprocessing, to answer A-decision(Q, ε, ρ) in $O\left(\frac{cm}{\varepsilon} \cdot \left(d \log \varepsilon^{-1} + \log n\right)\right)$ worst case time.*
  - *Improving the static expected $O(d^2 \cdot \frac{cn}{\sqrt{\varepsilon}} + d^2 \cdot cn \log n)$ algorithm by Bringmann and Künnemann [10, IJCGA'17]: saving a factor d and obtaining deterministic guarantees.*
- *For $\mathcal{X}$ an N-vertex planar graph under the shortest path metric, we can store P using $O(N^{1+o(1)})$ space and preprocessing, to answer A-decision(Q, ε, ρ) in $O\left(\frac{cm}{\varepsilon} \cdot \log^{2+o(1)} N\right)$ time.*
  - *Generalizing the static $O\left(N^{1+o(1)} + \frac{c \cdot m}{\varepsilon} \log^{2+o(1)} N\right)$ algorithm by Driemel, van der Hoog and Rotenberg [25, SoCG'22] to a data structure.*
- *For $\mathcal{X}$ an N-vertex graph under the shortest path metric, we can fix ε and store P using $O(\frac{N}{\varepsilon} \log N)$ space and preprocessing, to answer A-decision(Q, ε, ρ) in $O\left(\frac{cm}{\varepsilon} \cdot (\varepsilon^{-1} + \log n)\right)$ time.*
  - *Generalizing the static $O\left(N^{1+o(1)} + \frac{c \cdot m}{\varepsilon} \log^{2+o(1)} N\right)$ algorithm by Driemel, van der Hoog and Rotenberg [25, SoCG'22] to a data structure.*

Applying the 1-TADD. We can transform the decision variant of the Fréchet distance to the optimization variant, by using only a well-separated pair decomposition of P (mapped to $\mathbb{R}^1$) with itself. This allows us to answer the optimization variant for the discrete Fréchet distance without an exponential dependence on the dimension (speeding up even static algorithms). In full generality, combining Theorem 4 with Oracles 12 implies:

▶ **Corollary 7.** *Let P be a c-packed curve in a metric space $\mathcal{X}$.*

- *For $\mathcal{X} = \mathbb{R}^d$ in the real-RAM model, we can store P using $O(n)$ space and $O(n \log n)$ preprocessing, to answer A-value(Q, ε) in $O\left(\frac{cm}{\varepsilon} \log n (d + \log \frac{c \cdot m}{\varepsilon} + \log n\right)$ time.*
  - *Improving the static $O(2^d n + d \cdot \frac{cn}{\sqrt{\varepsilon}} + d^2 \cdot cn \log n)$ algorithm by Bringmann and Künnemann [10, IJCGA'17]: removing the exponential dependency on the dimension.*
  - *Improving the dynamic $O(\left(O(\frac{1}{\varepsilon})\right)^{dm} \log \frac{1}{\varepsilon})$ space solution with $\tilde{O}(m \cdot d)$ query time by Filtser and Filtser [27, SODA'21]: allowing Q to have arbitrary length, and using linear as opposed to exponential space and preprocessing time. This applies only when P is c-packed.*
- *For $\mathcal{X} = \mathbb{R}^d$ under the L_2 metric in word-RAM, we can store P using $O(n)$ space and $O(n \log n)$ preprocessing, to answer A-value(Q, ε) in $O\left(\frac{cm}{\varepsilon} \log n (d \log n + \log \frac{c \cdot m}{\varepsilon}\right)$ time.*
  - *Improving upon the static expected $O(2^d n + d^2 \cdot \frac{cn}{\sqrt{\varepsilon}} + d^3 \cdot cn \log n)$ algorithm by Bringmann and Künnemann [10, IJCGA'17]: saving a factor $d2^d$ with deterministic guarantees.*
- *For $\mathcal{X}$ an N-vertex planar graph under the SP metric, we can store P using $O(N^{1+o(1)})$ space and preprocessing, to answer A-value(Q, ε) in $O\left(\frac{c \cdot m}{\varepsilon} \log n \cdot (\log^{2+o(1)} N + \log \frac{c \cdot m}{\varepsilon})\right)$ time.*
  - *Improving the static $O\left(N^{1+o(1)} + |E| \log |E| + \frac{c \cdot m}{\varepsilon} \log^{2+o(1)} N \log |E|\right)$ algorithm by Driemel, van der Hoog and Rotenberg [25, SoCG'22]: making it a data structure.*

- *For $\mathcal{X}$ an N-vertex graph under the SP metric, we can fix ε and store P using $O(\frac{N}{\varepsilon}\log N)$ space and preprocessing, to answer A-value(Q,ε) in $O\left(\frac{cm}{\varepsilon}\cdot\log n\cdot(\varepsilon^{-1}+\log\frac{c\cdot m}{\varepsilon}+\log n)\right)$ time.*
  - *Same improvement as above, except that this result is not adaptive to ε.*
- *For $\mathcal{X}$ an N-vertex simple polygon under geodesics, we can store P using $O(N\log N + n)$ space and preprocessing, to answer A-value(Q,ε) in $O\left(\frac{cm}{\varepsilon}\cdot\log n\cdot(\log N+\log\frac{c\cdot m}{\varepsilon}+\log n)\right)$ time.*
  - *No realism-parameter algorithm was known in this setting, because no TADD can be computed in this setting.*

We briefly note that all our results are also immediately applicable to subcurve queries:

▶ **Corollary 8.** *All results obtained in Section 6 can answer the subcurve variants of the A-decision and A-value queries for any $i,j\in[n]$ at no additional cost.*

3 Simplification and a data structure

To facilitate computations in arbitrary geodesic metric spaces, we modify the definition of μ-simplifications. Our modified definition has the same theoretical guarantees as the previous definition, but works in arbitrary metrics. Formally, we say that henceforth the μ-simplification is a curve obtained by starting with p_1, and recursively adding the first p_j such that the *length* of the subtrajectory $\ell(P[i,j])>\mu$, where p_i is the last vertex added to the simplified curve. This way, our μ-simplifications (and their computation) are independent of the ambient space and only depend on the edge lengths.
We construct a data structure such that for any value μ, we can efficiently obtain P^μ:

▶ **Definition 9.** *For any curve P in $\mathcal{X}$ with n vertices, for each $1<i\leq n$ we create a half-open interval $(\ell(P[1,i-1]),\ell(P[1,i])]$ in $\mathbb{R}^1$. This results in an ordered set of $O(n)$ disjoint intervals on which we build a balanced binary tree in $O(n)$ time.*

Our new definition and data structure allow us to obtain P^μ at query time:

▶ **Lemma 10.** *Let $P=(p_1,\ldots,p_n)$ be a curve in $\mathcal{X}$ stored in the data structure of Definition 9. For any value $\mu\geq 0$, any pair (i,j) with $i<j$, and any integer N we can report the first N vertices of the discrete μ-simplification $P[i,j]^\mu$ in $O(N\log n)$ time.*

Proof. The first vertex of $P[i,j]^\mu$ is p_i. We inductively add subsequent vertices. Suppose that we just added p_x to our output. We choose the value $a=\ell(P[1,x])+\mu$. We binary search in $O(\log n)$ time for the point p_y where the interval $(\ell(P[1,y-1]),\ell(P[1,y])]$ contains a. Per definition: the length $\ell(P[x,y])\geq\mu$. Moreover, for all $z\in(x,y)$ the length $\ell(P[x,z])<\mu$. Thus, p_y is the successor of p_x and we recurse if necessary. ◀

4 The 1-TADD technique

Let P and Q be curves in $\mathbb{R}^d$ under the Euclidean metric. In [22], they show an algorithm that (given the μ-simplified curves P^μ and Q^μ for $\mu=\varepsilon\rho/4$) they can decide whether $D_{\mathcal{F}}(P,Q)>\rho$ or $D_{\mathcal{F}}(P,Q)\leq(1+\varepsilon)\rho$ in $O(d\frac{c(n+m)}{\varepsilon})$ time. They then compute a $(1+\varepsilon)$-approximation of $D_{\mathcal{F}}(P,Q)$ through a binary search over TADD$(P,\ Q)$. This approach scales poorly with the dimension d because computing TADD$(P,\ Q)$ has an exponential dependency on d. We alleviate this through our 1-TADD definiton:

▶ **Definition 11.** *Given P, map each vertex p_i to $\lambda_i = \ell(P[1,i])$. Denote by $\Lambda = \{\lambda_i\}_{i=1}^{n}$. We define 1-TADD(P) as TADD(Λ, Λ).*

Our 1-TADD can be computed in $O(n \log n)$ time using $O(n)$ space [22].

▶ **Theorem 2.** *We can preprocess a pair of c-packed curves (P, Q) in $\mathbb{R}^d$ under any L_p metric with $|P| = n \geq |Q| = m$ in $O(n \log n)$ time s.t.: given any ε and an exact distance oracle, we can compute a $(1+\varepsilon)$-approximation of $D_{\mathcal{F}}(P, Q)$ in $O(d\frac{c(n+m)}{\varepsilon} \cdot (\log n + \log \varepsilon^{-1}))$ time.*

Proof. With slight abuse of notation, we say that $\mathrm{A}(P, Q, \varepsilon, \rho)$ is an algorithm that takes as input $P^{\varepsilon\rho/4}$ and $Q^{\varepsilon\rho/4}$ and outputs either $D_{\mathcal{F}}(P, Q) > \rho$ or $D_{\mathcal{F}}(P, Q) \leq (1+\varepsilon)\rho$. We briefly note given our new definition of μ-simplification, [22] present an $\mathrm{A}(P, Q, \varepsilon, \rho)$ algorithm with a runtime of $O(d\frac{c(n+m)}{\varepsilon})$ under any L_p metric (as Lemma 4.4 in [22] immediately works for our simplification definition). We use $\mathrm{A}(P, Q, \varepsilon, \rho)$ to approximate $D_{\mathcal{F}}(P, Q)$.

We preprocess P and Q by computing T_P = 1-TADD(P) and T_Q = 1-TADD(Q). We denote by I the set of intervals obtained by taking for each $a \in T_P \cup T_Q$ the interval $[4\varepsilon^{-1}a, 8\varepsilon^{-1}a]$ (we add the interval $[0, 0]$ to I). Given $\mathrm{A}(P, Q, \varepsilon, \rho)$ that runs in $O(d\frac{c(n+m)}{\varepsilon})$ time, we do binary search over I. Specifically, we iteratively select an interval $[a, b] \in I$ and run $\mathrm{A}(P, Q, \varepsilon, \rho)$ for ρ equal to either endpoint.

Note that for each ρ, we may use Lemma 10 to obtain both the $\varepsilon\rho/4$-simplifications of P and Q in $O((n+m)\log(n+m))$ time – which are required as input for $\mathrm{A}(P, Q, \varepsilon, \rho)$. We need to do this procedure at most $O(\log n)$ times before we reach one of two cases:

- Case 1: There exists $[a, b] \in I$, such that $D_{\mathcal{F}}(P, Q) > a$ and $D_{\mathcal{F}}(P, Q) < (1+\varepsilon)b$. We note that by definition of I, the values a and b differ by a factor 2. Thus, we may discretize the interval $[a, b]$ into $O(\varepsilon^{-1})$ points that are each at most $\frac{\varepsilon \cdot a}{2}$ apart (note that we implicitly discretize this interval, as an explicit discretization takes ε^{-1} time). By performing binary search over this discretized set, we report a $(1+\varepsilon)$-approximation of $D_{\mathcal{F}}(P, Q)$ by using $\mathrm{A}(P, Q, \varepsilon, \rho)$ at most $O(\log \varepsilon^{-1})$ times.
- Case 2: There exists no $[x, y] \in I$ such that $D_{\mathcal{F}}(P, Q) > x$ and $D_{\mathcal{F}}(P, Q) < (1+\varepsilon)y$.
  - Denote by $[a_{max}, b_{max}]$ the right-most interval in I. Consider the special case where $D_{\mathcal{F}}(P, Q) > b_{max}$. Since $b_{max} \geq \ell(P), \ell(Q)$ it follows that all $\rho > b_{max}$, P^{μ} and Q^{μ} for $\mu = \varepsilon\rho/4$ are an edge. Computing $D_{\mathcal{F}}(P^{\mu}, Q^{\mu})$ can therefore be done in $O(d)$ time which gives a $(1+\varepsilon)$-approximation of $D_{\mathcal{F}}(P, Q)$.

  If the special case does not apply then there exist two intervals $[a, b], [e, f] \in I$ such that $D_{\mathcal{F}}(P, Q) > b$ and $D_{\mathcal{F}}(P, Q) \leq (1+\varepsilon)e$, such that there exists no interval $[x, y] \in I$ that intersects $[b, e]$. We claim that all $\rho_1, \rho_2 \in [b, e]$: $P^{\varepsilon\rho_1/4} = P^{\varepsilon\rho_2/4}$ and $Q^{\varepsilon\rho_1/4} = Q^{\varepsilon\rho_2/4}$.
  - Indeed, suppose for the sake of contradiction that $P^{\varepsilon\rho_1/4} \neq P^{\varepsilon\rho_2/4}$. Let $\rho_1 < \rho_2$ and choose without loss of generality the smallest ρ_2 for which this is the case. Then there must exist a pair $p_i, p_j \in P$ where $\ell(P[i, j]) = \varepsilon\rho_2/4$. However, the distance $\ell(P[i, j])$ is the distance between λ_i and λ_j in the curve Λ and so there exist $a', b' \in T_P$ with $a' \leq \ell(P[i, j]) \leq b' \leq 2a'$. It follows that $\rho_2 = 4\varepsilon^{-1}\ell(P[i, j])$ lies in an interval in I which is a contradiction with the assumption that $\rho_1, \rho_2 \in [b, e]$.

  We choose $\rho = e$. Denote by X the set of reachable cells the Free Space Diagram of $(P^{\varepsilon\rho/4}, Q^{\varepsilon\rho/4}, \rho^*)$. The set X contains $O(\frac{c(n+m)}{\varepsilon})$ cells [22, Lemma 4.4]. It follows that there are $O(\frac{c(n+m)}{\varepsilon})$ values ρ' for which the reachability of X changes. We compute and sort these to get a sorted set R.

  Suppose for some $\rho' \in [b, \rho]$ that $D_{\mathcal{F}}(P\text{'}Q) \leq \rho'$. Denote by F an xy-monotone path in the Free Space Diagram of $(P^{\varepsilon\rho'/4}, Q^{\varepsilon\rho'/4}, \rho') = (P^{\varepsilon\rho/4}, Q^{\varepsilon\rho/4}, \rho')$. Per definition, F lies within X. Thus, we may binary search over the set $R \cap [b, \rho]$ (applying the ε-approximate decider at every step) to compute a $(1+\varepsilon)$-approximation of $D_{\mathcal{F}}(P, Q)$. ◀

5 Approximate distance oracles under the discrete Fréchet distance

We want to approximate $D_{\mathbb{F}}(P, Q)$ for curves P and Q that live in any geodesic ambient space $\mathcal{X}$. In most ambient spaces we do not have access to efficient exact distance oracles. In many ambient spaces however, it is possible to compute for any $\alpha > 0$ some $(1+\alpha)$-approximate distance oracle. This is a data structure $\mathcal{D}_{\mathcal{X}}^{\alpha}$ that for any two $a, b \in \mathcal{X}$ can report a value $d^{\circ}(a, b) \in [(1-\alpha)d(a, b), (1+\alpha)d(a, b)]$ in $O(T_\alpha)$ time. To distinguish between inaccuracy as a result of our algorithm and as a result of our oracle, we refer to $d^{\circ}(a, b)$ as the *perceived value* (as opposed to an approximate value).

▶ **Oracles 12.** *We present some examples of approximate distance oracles:*

- *For $\mathcal{X} \subseteq \mathbb{R}^d$ under the L_1, L_2, L_∞ metric in real-RAM we can compute the exact $d(a, b)$ in $O(d)$ time. Thus, for any α, we have an oracle $\mathcal{D}_{\mathcal{X}}^{\alpha}$ with $T_\alpha = O(d)$ query time.*
- *For $\mathcal{X} \subseteq \mathbb{R}^d$ under the L_2 metric executed in word-RAM, we can compute $d(a, b)$ in $O(d^2)$ expected time. Thus, we have an oracle $\mathcal{D}_{\mathcal{X}}^{\alpha}$ with $O(d^2)$ expected query time.*
- *For any $\mathcal{X} \subseteq \mathbb{R}^d$ under the L_2 metric in word-RAM, we can $(1+\alpha)$-approximate the distance between two points in $T_\alpha = O(d \log \alpha^{-1})$ worst case time using Taylor expansions.*
- *For $\mathcal{X}$ a planar weighted graph, Long and Pettie [36] store $\mathcal{X}$ with N vertices using $O\left(N^{1+o(1)}\right)$ space, to answer exact distance queries in $O\left((\log(N))^{2+o(1)}\right)$ time.*
- *For $\mathcal{X}$ as an arbitrary weighted graph, Thorup [43] compute a $(1+\alpha)$-approximate distance oracle in $O(N/\alpha \log N)$ time and space, and with a query-time of $O(1/\alpha)$.*
- *For $\mathcal{X}$ a simple N-vertex polygon, Guibas and Hershberger [32] store $\mathcal{X}$ in $O(N \log N)$ time in linear space, and answer exact geodesic distance queries in $O((\log N)$ time.*

We prove that we may approximately decide the Fréchet distance between P and Q using a $(1+\alpha)$-approximate distance oracle (for the discrete Fréchet distance).

▶ **Lemma 3.** *For any $\rho > 0$ and $0 < \varepsilon < 1$, choose $\rho^* = (1 + \frac{1}{2}\varepsilon)\rho$ and $\mu \leq \frac{1}{6}\varepsilon\rho$. Let $\mathcal{X}$ be any geodesic metric space and $\mathcal{D}_{\mathcal{X}}^{\varepsilon/6}$ be a $(1 + \frac{1}{6}\varepsilon)$-approximate distance oracle. For any curve $P = (p_1, \ldots, p_n)$ in $\mathcal{X}$ and any curve $Q = (q_1, \ldots, q_m)$ in $\mathcal{X}$:*

- *If for the discrete Fréchet distance, ${D_{\mathbb{F}}}^{\circ}(P^\mu, Q) \leq \rho^*$ then $D_{\mathbb{F}}(P, Q) \leq (1+\varepsilon)\rho$.*
- *If for the discrete Fréchet distance, ${D_{\mathbb{F}}}^{\circ}(P^\mu, Q) > \rho^*$ then $D_{\mathbb{F}}(P, Q) > \rho$.*

Proof. Per definition of $\mathcal{D}_{\mathcal{X}}^{\varepsilon/6}$: $\forall (p, q) \in P \times Q$, $d^{\circ}(p, q) \in [(1 - \frac{1}{6}\varepsilon)d(p, q), (1 + \frac{1}{6}\varepsilon)d(p, q)]$. It follows from $0 < \varepsilon < 1$ that:

$$\forall (p, q) \in P \times Q : \quad d(p, q) \leq \left(1 + \frac{1.1}{6}\varepsilon\right) d^{\circ}(p, q) \quad \wedge \quad d^{\circ}(p, q) \leq \left(1 + \frac{1}{6}\varepsilon\right) d(p, q).$$

Suppose that ${D_{\mathbb{F}}}^{\circ}(P^\mu, Q) \leq \rho^*$. There exists a (monotone) discrete walk F through $P^\mu \times Q$ such that for each $(i, j) \in F$: $d^{\circ}(P^\mu[i], q_j) \leq \rho^* = (1 + \frac{1}{2}\varepsilon)\rho$. It follows that:

$$d(P^\mu[i], q_j) \leq \left(1 + \frac{1.1}{6}\varepsilon\right) d^{\circ}(P^\mu[i], q_j) \leq \left(1 + \frac{1.1}{6}\varepsilon\right)\left(1 + \frac{1}{2}\varepsilon\right)\rho \leq \left(1 + \frac{5}{6}\varepsilon\right)\rho.$$

We will prove that this implies $D_{\mathbb{F}}(P, Q) \leq (1+\varepsilon)\rho$. We use F to construct a discrete walk F' through $P \times Q$. For each consecutive pair $(a, b), (c, d) \in F$ note that since F is a discrete walk, $P^\mu[a]$ and $P^\mu[c]$ are either the same vertex or incident vertices on P^μ. Denote by P_{ac} the vertices of P in between $P^\mu[a]$ and $P^\mu[c]$. It follows that:

$$\forall p' \in P_{ac} : \quad d(p', q_b) \leq d\left(P^\mu[a], q_b\right) + \mu \leq (1 + \frac{5}{6}\varepsilon)\rho + \frac{1}{6}\varepsilon\rho = (1+\varepsilon)\rho.$$

Now consider the following sequence of pairs of points:

$L_{ac} = (P^\mu[a], q_b) \cup \{(p', q_b) \mid p' \in P_{ac}\} \cup (P^\mu[c], q_d)$. We add the lattice points corresponding to L_{ac} to F'. It follows that we create a discrete walk F' in the lattice $|P| \times |Q|$ where for each $(i, j) \in F'$: $d(p_i, q_j) \leq (1 + \varepsilon)\rho$. Thus, $D_{\mathbb{F}}(P, Q) \leq (1 + \varepsilon)\rho$.

Suppose otherwise that $D_{\mathbb{F}}(P, Q) \leq \rho$. We will prove that $D_{\mathbb{F}}{}^\circ(P^\mu, Q) \leq \rho^*$. Indeed, consider a discrete walk F' in the lattice $|P| \times |Q|$ where for each $(i, j) \in F'$: $d(p_i, q_j) \leq \rho$. We construct a discrete walk F in $|P^\mu| \times |Q|$. Consider each $(i, j) \in F'$, If $p_i = P^\mu[a]$ for some integer a, we add (a, j) to F. Otherwise, denote by $P^\mu[a]$ the last vertex on P^μ that precedes p_i: we add (a, j) to F. Note that per definition of μ-simplification, $d(P^\mu[a], q_j) \leq d(p_i, q_j) + \mu \leq (1 + \frac{1}{6}\varepsilon)\rho$. It follows from the definition of our approximate distance oracle that $d^\circ(P^\mu[a], q_j) \leq (1 + \frac{1}{6}\varepsilon)(1 + \frac{1}{6}\varepsilon)\rho < (1 + \frac{1}{2}\varepsilon)\rho = \rho^*$. Thus, we may conclude that $D_{\mathbb{F}}{}^\circ(P^\mu, Q) \leq \rho^*$. ◀

6 Approximate Discrete Fréchet distance

We denote by $\mathcal{D}_{\mathcal{X}}^{\alpha}$ a $(1 + \alpha)$-approximate distance oracle over the geodesic metric space $\mathcal{X}$. Our input is some curve $P = (p_1, \ldots, p_n)$ in $\mathcal{X}$ which is c-packed in $\mathcal{X}$. We preprocess P to:

- answer A-decision(Q, ε, ρ) for any curve $Q = (q_1, \ldots, q_m)$, $\rho > 0$ and $0 < \varepsilon < 1$,
- answer A-value(Q, ε) for any curve $Q = (q_1, \ldots, q_m)$ and $0 < \varepsilon < 1$.

We obtain this result in four steps. In Section 5, we showed that we can answer A-decision(Q, ε, ρ) through comparing if the perceived Fréchet distance $D_{\mathbb{F}}{}^\circ(P^\mu, Q) \leq \rho^*$ for conveniently chosen μ and ρ^*. In Section 6.1 we define what we call the perceived free-space matrix. This is a $(0, 1)$-matrix $M_{\rho^*}^{A \times Q}$ for any two curves A and Q and any $\rho^* > 0$. We show that if A is the μ-simplified curve P^μ for some convenient μ, then the number of zeroes in $M_{\rho^*}^{P^\mu \times Q}$ is bounded.

In Section 6.2, we show a data structure that stores P to answer A-decision(Q, ε, ρ). We show how to cleverly navigate $M_{\rho^*}^{P^\mu \times Q}$ for conveniently chosen μ and ρ^*. The key insight in this new technique, is that we may steadily increase ρ^* whilst navigating the matrix. Finally, we extend this solution to answer A-value(Q, ε).

6.1 Perceived free space matrix and free space complexity

We define the perceived free space matrix to help answer A-decision queries. Given two curves (A, Q) and some ρ, we construct an $|A| \times |Q|$ matrix which we call the perceived free space matrix $M_\rho^{A \times Q}$. The i'th column corresponds to the i'th element $A[i]$ in A. We assign to each matrix cell $M_\rho^{P \times Q}[i, j]$ the integer 0 if $d^\circ(A[i], q_j) \leq \rho$ and integer 1 otherwise.

▶ **Observation 13.** *For all $\rho^* \geq 0$, and curves A and Q, the perceived discrete Fréchet distance $D_{\mathbb{F}}{}^\circ(A, Q)$ between A and Q is at most ρ^* if and only if there exists an (xy-monotone) discrete walk F from $(1, 1)$ to $(|A|, |Q|)$ where $\forall (i, j) \in F$, $M_{\rho^*}^{A \times Q}[i, j] = 0$.*

Computing $D_{\mathbb{F}}{}^\circ(P^\mu, Q)$. Previous results upper bound, for any choice of ρ, the number of zeroes in the FSM between $P^{\varepsilon\rho}$ and $Q^{\varepsilon\rho}$. We instead consider the perceived FSM, and introduce a new parameter $k \geq 1$ to enable approximate distance oracles. For any value ρ and some simplification value $\mu \geq \frac{\varepsilon\rho}{k}$, we upper bound the number of zeroes in the perceived FSM $M_{\rho^*}^{P^\mu \times Q}$ for the conveniently chosen $\rho^* = (1 + \frac{\varepsilon}{2})\rho$:

▶ **Lemma 14.** *Let $P = (p_1, \ldots, p_n)$ be a c-packed discrete curve in $\mathcal{X}$. For any $\rho > 0$ and $0 < \varepsilon < 1$, denote $\rho^* = (1 + \frac{\varepsilon}{2})\rho$. For any $k \geq 1$, denote by P^μ its μ-simplified curve for $\mu \geq \frac{\varepsilon\rho}{k}$. For any curve $Q = (q_1, \ldots, q_m) \subset \mathcal{X}$ the matrix $M_{\rho^*}^{P^\mu \times Q}$ contains at most $8 \cdot \frac{c \cdot k}{\varepsilon}$ zeroes per row.*

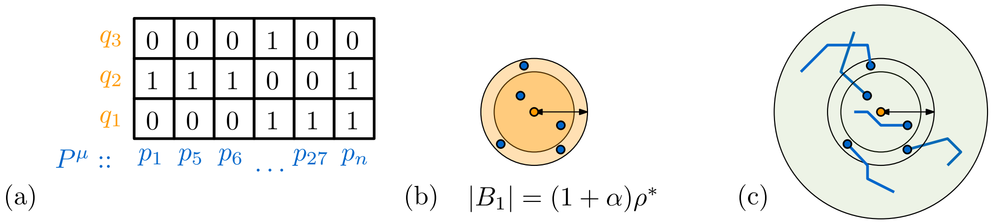

q_3 0 0 0 1 0 0
q_2 1 1 1 0 0 1
q_1 0 0 0 1 1 1
P^μ :: p_1 p_5 p_6 ... p_{27} p_n
(a)
(b) $|B_1| = (1+\alpha)\rho^*$
(c)

Figure 1 (a) For some value of μ, the μ-simplified curve is $(p_1, p_5, p_6, \ldots, p_{27}, p_n)$. We show the matrix $M_{\rho^*}^{P^\mu \times Q}$. (b) For the point q_3, we claim that there are more than $Z = 8\frac{ck}{\varepsilon}$ zeroes in its corresponding row. Thus, the ball B_1 with radius $(1+\alpha)\rho^*$ contains more than Z points. (c) For each of these points, there is a unique segment along P contained in the ball B_2.

Proof. The proof is by contradiction. Suppose that the j'th row of $M_{\rho^*}^{P^\mu \times Q}$ contains strictly more than $8 \cdot \frac{c \cdot k}{\varepsilon}$ zeroes. Let $P_0 \subset P^\mu$ be the vertices corresponding to these zeroes. Consider the ball B_1 centered at q_j with radius $|B_1| = (1+\alpha)\rho^*$ and the ball B_2 with radius $2|B_1|$ (Figure 1). Each $p_i \in P_0$ must be contained in B_1 and thus $d(p_i, q_j) \leq (1+\alpha)\rho^*$. For each $p_i \in P_0$ denote by S_i the contiguous sequence of vertices of P^μ starting at p_i of length μ. Observe that since $\varepsilon < 1$: $S_i \subset B_2$. Per definition of simplification, each S_i are non-coinciding subcurves. This lower bounds $\ell(P \cap B_2)$:

$$\ell(P \cap B_2) \geq \sum_{p \in P_0} \ell(S_i) = \sum_{p \in P_0} \mu > 2(1+\alpha)(1+\frac{\varepsilon}{2}) \cdot \frac{c \cdot k}{\varepsilon} \cdot \mu \geq 2(1+\alpha) \cdot c \cdot \rho^* \geq c \cdot |B_2|,$$

where $2(1+\alpha)(1+\frac{\varepsilon}{2}) \cdot \frac{c \cdot k}{\varepsilon} < 8 \cdot \frac{c \cdot k}{\varepsilon}$ (since $\alpha < 1$ and $\varepsilon < 1$) – contradicting c-packedness. ◀

6.2 A data structure for answering A-decision(Q, ε, ρ)

We showed in Sections 5 and 6.1 that for any c-packed curve P, $\rho > 0$ and $0 < \varepsilon < 1$ we can choose suitable values $\frac{\varepsilon\rho}{k} \leq \mu \leq \frac{\varepsilon\rho}{6}$ to upper bound the number of zeroes in $M_{\rho^*}^{P^\mu \times Q}$. Moreover, for $\rho^* = (1 + \frac{1}{\varepsilon})\rho$ we know that comparing $D_{\mathbb{F}}{}^{\circ}(P^\mu, Q) \leq \rho^*$ implies an answer to A-decision(Q, ε, ρ).

We now define a data structure, so that for any μ and any (i, j) we can report the μ-simplification of $P[i, j]$ in $O(|P[i, j]|)$ time. We use this to answer the decision variant.

▶ **Theorem 4.** *Let $\mathcal{X}$ be any geodesic space and $\mathcal{D}_{\mathcal{X}}^{\alpha}$ be a $(1+\alpha)$-approximate distance oracle with $O(T_\alpha)$ query time. Let $P = (p_1, \ldots, p_n)$ be any c-packed curve in $\mathcal{X}$. We can store P using $O(n)$ space and preprocessing, such that for any curve $Q = (q_1, \ldots, q_m)$ in $\mathcal{X}$ and any $\rho > 0$ and $0 < \varepsilon < 1$, we can answer A-decision(Q, ε, ρ) for the discrete Fréchet distance in:*

$$O\left(\frac{c \cdot m}{\varepsilon} \cdot (T_{\varepsilon/6} + \log n)\right) \text{ time.}$$

Proof. We store P in the data structure of Definition 9 using $O(n)$ space and preprocessing time. Given a query A-decision(Q, ε, ρ) we choose $\alpha = \frac{1}{6}\varepsilon$, $\rho^* = (1 + \frac{1}{2}\varepsilon)$ and $\mu = \frac{\varepsilon\rho}{6}$. We test if $D_{\mathcal{F}}{}^{\circ}(P^\mu, Q) \leq \rho^*$. By Lemma 3, if $D_{\mathbb{F}}{}^{\circ}(P^\mu, Q) \leq \rho^*$ then $D_{\mathbb{F}}(P, Q) \leq (1+\varepsilon)\rho$ and otherwise $D_{\mathbb{F}}(P, Q) > \rho$. We consider the matrix $M_{\rho^*}^{P^\mu \times Q}$.

By Observation 13, $D_{\mathbb{F}}{}^{\circ}(P^\mu, Q) \leq \rho^*$ if and only if there exists a discrete walk F from $(1, 1)$ to $(|P^\mu|, |Q|)$ where for each $(i, j) \in F$: $M_{\rho^*}^{P^\mu \times Q}[i, j] = 0$. We will traverse this matrix in a depth-first manner as follows: starting from the cell $(1, 1)$, we test if $M_{\rho^*}^{P^\mu \times Q}[1, 1] = 0$. If so, we push $(1, 1)$ onto a stack. Each time we pop a tuple (i, j) from the stack, we inspect their $O(1)$ neighbors $\{(i+1, j), (i, j+1), (i+1, j+1)\}$. If $M_{\rho^*}^{P^\mu \times Q}[i', j'] = 0$, we push (i', j')

onto our stack. It takes $O(\log n)$ time to obtain the $i+1$'th vertex of P^μ, and $O(T_{\varepsilon/6})$ to determine the value of e.g., $M_{\rho^*}^{P^\mu \times Q}[i+1, j]$. Thus each time we pop the stack, we spend $O((T_{\varepsilon/6}) + \log n)$ time.

By Lemma 14 (noting $\varepsilon < 1$ and setting $k = 6$), we push at most $O(\frac{cm}{\varepsilon})$ tuples onto our stack. Therefore, we spend $O(\frac{cm}{\varepsilon}(T_{\varepsilon/6} + \log n))$ total time. By Observation 13, $D_{\mathbb{F}}{}^\circ(P^\mu, Q) \leq \rho^*$ if and only if we push $(|P^\mu|, |Q|)$ onto our stack. We test this in $O(1)$ additional time per operation. Thus, the theorem follows. ◀

6.3 A data structure for answering A-value(Q, ε)

Finally, we show how to answer the A-value(Q, ε, ρ) query. At this point, we could immediately apply Theorem 2 to answer A-value(Q, ε) at the cost of a factor $O(\log n + \log \varepsilon^{-1})$. However, for the discrete Fréchet distance we show that the factor $O(\log \varepsilon^{-1})$ can be avoided. To this end, we leverage the variable $k \geq 1$ introduced in the definition of $\mu \geq \frac{\varepsilon\rho}{k}$:

▶ **Theorem 5.** *Let $\mathcal{X}$ be a geodesic metric space and $\mathcal{D}_{\mathcal{X}}^{\alpha}$ be a $(1+\alpha)$-approximate distance with $O(T_\alpha)$ query time. Let $P = (p_1, \ldots, p_n)$ be any c-packed curve in $\mathcal{X}$. We can store P using $O(n)$ space and $O(n \log n)$ preprocessing time, such that for any curve $Q = (q_1, \ldots, q_m)$ in $\mathcal{X}$ and any $0 < \varepsilon < 1$, we can answer A-Value(Q, ε) for the discrete Fréchet distance in:*

$$O\left(\frac{c \cdot m}{\varepsilon} \cdot \log n \cdot \left(T_{\varepsilon/6} + \log \frac{c \cdot m}{\varepsilon} + \log n\right)\right) \text{ time.}$$

Proof. We preprocess P using Lemma 10 in $O(n)$ space and time. We store P in the data structure of Definition 11. This way, we obtain $T = \text{TADD}(\Lambda, \Lambda)$ where Λ is the curve P mapped to $\mathbb{R}^1$. We denote for all $s \in T$ by $I_s = [c_s, 2 \cdot c_s]$ the corresponding interval and obtain a sorted set of intervals $\mathcal{I} = \{I_s\}$.

Given a query (Q, ε), we set $\alpha \leftarrow \varepsilon/6$ and obtain $\mathcal{D}_{\mathcal{X}}^{\alpha}$. We (implicitly) rescale each interval $I_i \in \mathcal{I}$ by a factor $\frac{6}{\varepsilon}$, creating for I_s the interval $I_s^\varepsilon = [\frac{6 \cdot c_s}{\varepsilon}, \frac{12 \cdot c_s}{\varepsilon}]$. This creates a sorted set $\mathcal{I}^\varepsilon$ of pairwise disjoint intervals. Intuitively, these are the intervals over $\mathbb{R}^1$ where for $\rho \in I_s^\varepsilon$, the μ-simplification P^μ for $\mu = \frac{\varepsilon\rho}{6}$ may change.

We binary search over $\mathcal{I}^\varepsilon$. For each boundary point λ of an interval I_s^ε we query A-decision$(Q, \varepsilon, \lambda)$: discarding half of the remaining intervals in $\mathcal{I}^\varepsilon$. It follows that in $O(\frac{c \cdot m}{\varepsilon} \cdot \log n \cdot (T_{\varepsilon/6} + \log n))$ time, we obtain one of two things:

a) an interval I_s^ε where $\exists \rho^* \in I_s^\varepsilon$ that is a $(1+\varepsilon)$-approximation of $D_{\mathbb{F}}(P, Q)$, or

b) a maximal interval I^* disjoint of the intervals in $\mathcal{I}^\varepsilon$ where $\exists \rho^* \in I^*$ that is a $(1+\varepsilon)$-approximation of $D_{\mathbb{F}}(P, Q)$.

Denote by λ the left boundary of I_s^ε or I^*: it lower bounds $D_{\mathbb{F}}(P, Q)$. Note that if I^* precedes all of $\mathcal{I}^\varepsilon$, $\lambda = 0$. We now compute a $(1+\varepsilon)$-approximation of $D_{\mathbb{F}}(P, Q)$ as follows:

FindApproximation(λ).

```
1. Compute $C = \frac{d^\circ(p_1, q_1)}{(1+\frac{1}{2}\varepsilon)}$.
2. Initialize $\rho^* \leftarrow (1+\frac{1}{2}\varepsilon) \cdot \max\{C, \lambda\}$ and set a constant $\mu \leftarrow \frac{\varepsilon}{6} \cdot \lambda$.
3. Push the lattice point $(1, 1)$ onto a stack.
4. Whilst the stack is not empty do:
   - Pop a point $(i, j)$ and consider the $O(1)$ neighbors $(p_a, q_b)$ of $(p_i, q_j)$ in $M_{\rho^*}^{P^\mu \times Q}$:
      - If $d^\circ(p_a, q_b) \leq \rho^*$, push $(a, b)$ onto the stack.
      - Else, store $d^\circ(p_a, q_b)$ in a min-heap.
   - If we push $(p_n, q_m)$ onto the stack do:
      - Output $\nu = \frac{\rho^*}{(1+\frac{1}{2}\varepsilon)}$.
5. If the stack is empty, we extract the minimal $d^\circ(p_a, p_b)$ from the min-heap.
   - Update $\rho^* \leftarrow (1+\frac{1}{2}\varepsilon) \cdot d^\circ(p_a, q_b)$, push $(a, b)$ onto the stack and go to line 4.
```

Correctness. Suppose that our algorithm pushes (p_n, q_m) onto the stack and let at this time of the algorithm, $\rho^* = (1 + \frac{1}{2}\varepsilon)\nu$. Per definition of the algorithm, $\nu \geq \lambda$ is the minimal value for which the matrix $M_{\rho^*}^{P^\mu \times Q}$ contains a walk F from $(1,1)$ to (n,m) where for each $(i,j) \in F$: $M_{\rho^*}^{P^\mu \times Q}[i,j] = 0$. Indeed, each time we increment ρ^* by the minimal value required to extend any walk in $M_{\rho^*}^{P^\mu \times Q}$. Moreover, we fixed $\mu \leftarrow \frac{\varepsilon}{6}\lambda$ and thus $\mu \leq \frac{\varepsilon}{6}\nu$. Thus we may apply Lemma 3 to defer that ν is the minimal value for which $D_{\mathbb{F}}(P,Q) \leq (1+\varepsilon)\nu$.

Running time. We established that the binary search over $\mathcal{I}^\varepsilon$ took $O(\frac{c \cdot m}{\varepsilon} \cdot \log n \cdot (T_{\varepsilon/6} + \log n))$ time. We upper bound the running time of our final routine. For each pair (p_i, q_j) that we push onto the stack we spend at most $O(T_{\varepsilon/6} + \log \frac{c \cdot m}{\varepsilon} + \log n)$ time as we:

- Obtain the $O(1)$ neighbors of (p_i, q_j) through our data structure in $O(\log n)$ time,
- Perform $O(1)$ distance oracle queries in $O(T_{\varepsilon/6})$ time, and
- Possibly insert $O(1)$ neighbors into a min-heap. The min-heap has size at most K: the number of elements we push onto the stack. Thus, this takes $O(\log K)$ insertion time.

What remains is to upper bound the number of items we push onto the stack. Note that we only push an element onto the stack, if for the current value ρ^* the matrix $M_{\rho^*}^{P^\mu \times Q}$ contains a zero in the corresponding cell. We now refer to our earlier case distinction.

Case (a): Since $\varepsilon < 1$ we know that $\rho^* \in [\lambda, 4 \cdot \lambda]$. We set $\mu = \frac{\varepsilon}{6}\lambda$. So $\mu \geq \frac{1}{k}\varepsilon\rho^*$ for $k = 24$. Thus, we may immediately apply Lemma 14 to conclude that we push at most $O(\frac{c \cdot m}{\varepsilon})$ elements onto the stack.

Case (b): Denote by $\gamma = \frac{\varepsilon}{6}\nu$. Per definition of our re-scaled intervals, the open interval (μ, γ) does not intersect with any interval in the non-scaled set $\mathcal{I}$. It follows that $P^\mu = P^\gamma$ and that for two consecutive vertices $p_i, p_l \in P^\mu$: $\ell(P[i,l]) > \gamma$. From here, we essentially redo Lemma 14 for this highly specialized setting. The proof is by contradiction, where we assume that for $\rho^* = (1 + \frac{\varepsilon}{2})\nu$ there are more than $8 \cdot 6 \cdot \frac{c}{\varepsilon}$ zeroes in the j'th row of $M_{\rho^*}^{P^\mu \times Q}$. Denote by $P_0 \subset P^\mu$ the vertices corresponding to these zeroes. We construct a ball B_1 centered at q_j with radius $2\rho^*$ and a ball B_2 with radius $2|B_1|$. We construct a subcurve S_i of P starting at $p_i \in P_0$ of length γ. The critical observation is, that our above analysis implies that all the subcurves S_i do not coincide (since each of them start with a vertex in P^μ). Since $\varepsilon < 1$, each segment S_i is contained in B_2. However, this implies that B_2 is not c-packed since: $\ell(P \cap B_2) \geq \sum_i \ell(S_i) = \sum_i \gamma > 8 \cdot 6 \frac{c}{\varepsilon}\gamma \geq 4 \cdot c \cdot \rho^* \geq 2 \cdot c \cdot |B_2|$. Thus, we always push at most $O(\frac{c \cdot m}{\varepsilon})$ elements onto our stack and this implies our running time. ◀

References

1 Pankaj K Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014. doi:10.1137/130920526.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995. doi:10.1142/S0218195995000064.

3 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004. doi:10.1007/S00453-003-1042-5.

4 Boris Aronov, Omrit Filtser, Michael Horton, Matthew J. Katz, and Khadijeh Sheikhan. Efficient nearest-neighbor query and clustering of planar curves. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R Salavatipour, editors, *Algorithms and Data Structures*, pages 28–42, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-24766-9_3.

5 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In Yossi Azar and Thomas Erlebach, editors, *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2006. doi:10.1007/11841036_8.

6 Alessandro Bombelli, Lluis Soler, Eric Trumbauer, and Kenneth D Mease. Strategic air traffic planning with Fréchet distance aggregation and rerouting. *Journal of Guidance, Control, and Dynamics*, 40(5):1117–1129, 2017.

7 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864, 2005. URL: `http://www.vldb.org/archives/website/2005/program/paper/fri/p853-brakatsoulas.pdf`.

8 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014. `doi:10.1109/FOCS.2014.76`.

9 Karl Bringmann, Anne Driemel, André Nusser, and Ioannis Psarros. Tight bounds for approximate near neighbor searching for time series under the fréchet distance. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 517–550. SIAM, 2022. `doi:10.1137/1.9781611977073.25`.

10 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(01n02):85–119, 2017. `doi:10.1142/S0218195917600056`.

11 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2017. `doi:10.1145/3139958.3139964`.

12 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. `doi:10.1007/S00454-017-9878-7`.

13 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. Seth says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2887–2901. SIAM, 2019. `doi:10.1137/1.9781611975482.179`.

14 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. `doi:10.1080/13658816.2019.1684498`.

15 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I Silveira, and Frank Staals. Efficient Fréchet distance queries for segments. *European Symposium on Algorithms*, 2022.

16 Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the fréchet distance. In Matthias Müller-Hannemann and Renato Fonseca F. Werneck, editors, *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2011, Holiday Inn San Francisco Golden Gateway, San Francisco, California, USA, January 22, 2011*, pages 75–83. SIAM, 2011. `doi:10.1137/1.9781611972917.8`.

17 Jacobus Conradi, Anne Driemel, and Benedikt Kolbe. Revisiting the fr\'echet distance between piecewise smooth curves. *arXiv preprint arXiv:2401.03339*, 2024. `doi:10.48550/arXiv.2401.03339`.

18 Mark De Berg, Atlas F Cook IV, and Joachim Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013. `doi:10.1016/J.COMGEO.2012.11.006`.

19 Mark de Berg, Ali D Mehrabi, and Tim Ophelders. Data structures for Fréchet queries in trajectory data. In *29th Canadian Conference on Computational Geometry (CCCG'17)*, pages 214–219, 2017.

20 Thomas Devogele. A new merging process for data integration based on the discrete Fréchet distance. In *Advances in spatial data handling*, pages 167–181. Springer, 2002.

21 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013. `doi:10.1137/120865112`.

22 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discret. Comput. Geom.*, 48(1):94–127, 2012. `doi:10.1007/s00454-012-9402-z`.

23 Anne Driemel and Ioannis Psarros. $(2+\epsilon)$-ANN for time series under the Fréchet distance. *Workshop on Algorithms and Data structures (WADS)*, 2021.

24 Anne Driemel, Ioannis Psarros, and Melanie Schmidt. Sublinear data structures for short Fréchet queries. *CoRR*, abs/1907.04420, 2019. `arXiv:1907.04420`.

25 Anne Driemel, Ivor van der Hoog, and Eva Rotenberg. On the discrete Fréchet distance in a graph. In *International Symposium on Computational Geometry (SoCG 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

26 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

27 Arnold Filtser and Omrit Filtser. Static and streaming data structures for fréchet distance queries. In Dániel Marx, editor, *Symposium on Discrete Algorithms (SODA) 2021*, pages 1150–1170. SIAM, 2021. `doi:10.1137/1.9781611976465.71`.

28 Arnold Filtser, Omrit Filtser, and Matthew J. Katz. Approximate nearest neighbor for curves – simple, efficient, and deterministic. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:19, 2020. `doi:10.4230/LIPIcs.ICALP.2020.48`.

29 Omrit Filtser. Universal approximate simplification under the discrete fréchet distance. *Inf. Process. Lett.*, 132:22–27, 2018. `doi:10.1016/j.ipl.2017.10.002`.

30 Joachim Gudmundsson, Martin P. Seybold, and Sampson Wong. Map matching queries on realistic input graphs under the fréchet distance. *Symposium on Discrete Algorithms (SODA)*, 2023.

31 Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Fréchet distance queries in trajectory data. In *The Third Iranian Conference on Computational Geometry (ICCG 2020)*, pages 29–32, 2020.

32 Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. In *Symposium on Computational geometry (SoCG)*, 1987.

33 Sariel Har-Peled. *Geometric approximation algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Soc., 2011.

34 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of bioinformatics and computational biology*, 6(01):51–64, 2008. `doi:10.1142/S0219720008003278`.

35 Maximilian Konzack, Thomas McKetterick, Tim Ophelders, Maike Buchin, Luca Giuggioli, Jed Long, Trisalyn Nelson, Michel A Westenberg, and Kevin Buchin. Visual analytics of delays and interaction in movement data. *International Journal of Geographical Information Science*, 31(2):320–345, 2017. `doi:10.1080/13658816.2016.1199806`.

36 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2517–2537. SIAM, 2021. `doi:10.1137/1.9781611976465.149`.

37 Ariane Mascret, Thomas Devogele, Iwan Le Berre, and Alain Hénaff. Coastline matching process based on the discrete Fréchet distance. In *Progress in Spatial Data Handling*, pages 383–400. Springer, 2006.

38 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983. `doi:10.1145/2157.322410`.

39 Otfried Schwarzkopf and Jules Vleugels. Range searching in low-density environments. *Inf. Process. Lett.*, 60(3):121–127, 1996. `doi:10.1016/S0020-0190(96)00154-8`.

40 Roniel S. De Sousa, Azzedine Boukerche, and Antonio A. F. Loureiro. Vehicle trajectory similarity: Models, methods, and applications. *ACM Comput. Surv.*, 53(5), September 2020. doi:10.1145/3406096.

41 E Sriraghavendra, K Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 461–465. IEEE, 2007. doi:10.1109/ICDAR.2007.4378752.

42 Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32, 2020. doi:10.1007/S00778-019-00574-9.

43 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, 2004. doi:10.1145/1039488.1039493.

44 Ivor van der Hoog, Eva Rotenberg, and Sampson Wong. Data structures for approximate discrete Fréchet distance. *CoRR*, abs/2212.07124, 2022. doi:10.48550/arXiv.2212.07124.

45 A. Frank van der Stappen. *Motion planning amidst fat obstacles.* University Utrecht, 1994.

46 Rene Van Oostrum and Remco Veltkamp. Parametric search made practical. In *Symposium on Computational Geometry (C)*, pages 1–9, 2002.

47 Dong Xie, Feifei Li, and Jeff M Phillips. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment*, 10(11):1478–1489, 2017. doi:10.14778/3137628.3137655.

48 Daming Xu. *Well-separated pair decompositions for doubling metric spaces.* PhD thesis, Carleton University, 2005.

Constant Approximating Disjoint Paths on Acyclic Digraphs Is W[1]-Hard

Michał Włodarczyk
University of Warsaw, Poland

Abstract

In the Disjoint Paths problem, one is given a graph with a set of k vertex pairs (s_i, t_i) and the task is to connect each s_i to t_i with a path, so that the k paths are pairwise disjoint. In the optimization variant, Max Disjoint Paths, the goal is to maximize the number of vertex pairs to be connected. We study this problem on acyclic directed graphs, where Disjoint Paths is known to be W[1]-hard when parameterized by k. We show that in this setting Max Disjoint Paths is W[1]-hard to c-approximate for any constant c. To the best of our knowledge, this is the first non-trivial result regarding the parameterized approximation for Max Disjoint Paths with respect to the natural parameter k. Our proof is based on an elementary self-reduction that is guided by a certain combinatorial object constructed by the probabilistic method.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractability, hardness of approximation, disjoint paths

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.57

Funding *Michał Włodarczyk*: Supported by Polish National Science Centre SONATA-19 grant number 2023/51/D/ST6/00155.

1 Introduction

The Disjoint Paths problem has attracted a lot of attention both from the perspective of graph theory and applications [23, 46, 50, 52]. Both decision variants, where one requires the paths to be either vertex-disjoint or edge-disjoint, are known to be NP-hard already on very simple graph classes [27, 37, 44, 45]. This has motivated the study of Disjoint Paths through the lens of parameterized complexity. Here, the aim is to develop algorithms with a running time of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where f is some computable function of a parameter k and n is the input size. A problem admitting such an algorithm is called *fixed-parameter tractable* (FPT). In our setting, k is the number of vertex pairs to be connected. On undirected graphs, both variants of Disjoint Paths have been classified as FPT thanks to the famous Graph Minors project by Robertson and Seymour [49] (see [32, 36] for later improvements). This was followed by a line of research devoted to designing faster FPT algorithms on planar graphs [1, 13, 41, 48, 54].

On directed graphs, there is a simple polynomial transformation between the vertex-disjoint and the edge-disjoint variants, so these two problems turn out equivalent. Here, the problem becomes significantly harder: It is already NP-hard for $k = 2$ [22]. The situation is slightly better for acyclic digraphs (DAGs) where Disjoint Paths can be solved in time $n^{\mathcal{O}(k)}$ [22] but it is W[1]-hard [51] (cf. [2]) hence unlikely to be FPT. In addition, no $n^{o(k)}$-time algorithm exists under the assumption of the Exponential Time Hypothesis (ETH) [12]. Very recently, it has been announced that Disjoint Paths is FPT on Eulerian digraphs [5]. It is also noteworthy that the vertex-disjoint and edge-disjoint variants are not equivalent on planar digraphs as the aforementioned reduction does not preserve planarity. Indeed, here the vertex-disjoint version is FPT [17] whereas the edge-disjoint version is W[1]-hard [12].

In the optimization variant, called Max Disjoint Paths, we want to maximize the number of terminals pairs connected by disjoint paths. The approximation status of this problem has been studied on various graph classes [8, 10, 15, 14, 20, 34, 35]. On acyclic

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 57; pp. 57:1–57:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

digraphs the best approximation factor is $\mathcal{O}(\sqrt{n})$ [9] and this cannot be improved unless P=NP [7]. A different relaxation is to allow the algorithm to output a solution in which every vertex appears in at most c paths (or to conclude that there is no vertex-disjoint solution). Kawarabayashi, Kobayashi, and Kreutze [31] used the directed half-integral grid theorem to design a polynomial-time algorithm for directed Disjoint Paths with congestion $c = 4$ for every k. In other words, such a relaxed problem belongs to the class XP. Subsequently, the congestion factor has been improved to $c = 3$ [33] and $c = 2$ [24].

Hardness of FPT approximation. For problems that are hard from the perspective of both approximation and FPT algorithms, it is natural to exploit the combined power of both paradigms and consider FPT approximation algorithms. Some prominent examples are an FPT approximation scheme for k-Cut [43] and an FPT 2-approximation for Directed Odd Cycle Transversal [42] parameterized by the solution size k. However, several important problems proved to be resistant to FPT approximation as well. The first hardness results in this paradigm have been obtained under a relatively strong hypothesis, called Gap-ETH [6]. Subsequently, an $\mathcal{O}(1)$-approximation for k-Clique was shown to be W[1]-hard [39] and later the hardness bar was raised to $k^{o(1)}$ [29]. In turn, k-Dominating Set is W[1]-hard to $f(k)$-approximate for any function f [30] and W[2]-hard to $\mathcal{O}(1)$-approximate [40]. More results are discussed in the survey [21].

Proving approximation hardness under Gap-ETH is easier compared to the assumption FPT≠W[1] because Gap-ETH already assumes hardness of a problem with a *gap*. Indeed, relying just on FPT≠W[1] requires the reduction to perform some kind of *gap amplification*, alike in the PCP theorem [19]. Very recently, the so-called *Parameterized Inapproximability Hypothesis* (PIH) has been proven to follow from ETH [25]. This means that ETH implies FPT approximation hardness of Max 2-CSP parameterized by the number of variables within some constant approximation factor $c > 1$, which has been previously used as a starting point for parameterized reductions [4, 26, 42, 47]. It remains open whether PIH can be derived from the weaker assumption FPT≠W[1].

Lampis and Vasilakis [38] showed that undirected Max Vertex-Disjoint Paths admits an FPT approximation scheme when parameterized by *treedepth* but, assuming PIH, this is not possible under parameterization by *pathwdith*. See [11, 20] for more results on approximation for Max Disjoint Paths under structural parameterizations. Bentert, Fomin, and Golovach [3] considered the Max Vertex-Disjoint Shortest Paths problem where we additionaly require each path in a solution to be a shortest path between its endpoints. They ruled out FPT(k) approximation with factor $k^{o(1)}$ for this problem assuming FPT≠W[1] and with factor $o(k)$ assuming Gap-ETH.

Our contribution. We extend the result by Slivkins [51] by showing that Max Disjoint Paths on acyclic digraphs does not admit an FPT algorithm that is a q-approximation, for any constant q. We formulate our hardness result as W[1]-hardness of the task of distinguishing between instances that are fully solvable from those in which less than a $\frac{1}{q}$-fraction of the requests can be served at once. Since a q-approximation algorithm could be used to tell these two scenarios apart, the following result implies hardness of approximation. We refer to a pair (s_i, t_i) as a *request* that should be *served* by a path connecting s_i to t_i.

▶ **Theorem 1.** *Let $q \in \mathbb{N}$ be a constant. It is W[1]-hard to distinguish whether for a given instance of k-Dag Disjoint Paths:*

1. *all the requests can be served simultaneously, or*
2. *no set of k/q requests can be served simultaneously.*

Our proof is elementary and does not rely on coding theory or communication complexity as some previous W[1]-hardness of approximation proofs [30, 39]. Instead, we give a gap-amplifying self-reduction that is guided by a certain combinatorial object constructed via the probabilistic method.

Techniques. A similar parameterized gap amplification technique has been previously applied to the k-Steiner Orientation problem: given a graph G with both directed and undirected edges, together with a set of vertex pairs $(s_1, t_1), \dots, (s_k, t_k)$, we want to orient all the undirected edges in G to maximize the number of pairs (s_i, t_i) for which t_i is reachable from s_i. The problem is W[1]-hard and the gap amplification technique can be used to establish W[1]-hardness of constant approximation [53]. The idea is to create multiple copies of the original instance and connect them sequentially into many layers, in such a way that the fraction of satisfiable requests decreases as the number of layers grows. What distinguishes k-Steiner Orientation from our setting though is that therein we do not require the (s_i, t_i)-paths to be disjoint. So it is allowed to make multiple copies of each request (s_i, t_i) and connect the t_i-vertices to the s_i-vertices in the next layer in one-to-many fashion. Such a construction obviously cannot work for Dag Disjoint Paths. Instead, will we construct a combinatorial object yielding a scheme of connections between the copies of the original instance, with just one-to-one relation between the terminals from the consecutive layers.

Imagine a following construction: given an instance I of k-Dag Disjoint Paths we create $2k$ copies of I: $I^1_1, \dots, I^1_k$ and $I^2_1, \dots, I^2_k$. Next, for each $i \in [k]$ we choose some permutation $\pi_i \colon [k] \to [k]$ and for each $j \in [k]$ we connect the sink t_j in I^1_i to the source s_i in $I^2_{\pi_i(j)}$. See Figure 1 on page 6. Then for each $(i, j) \in [k]^2$ we request a path from the source s_j in I^1_i to the sink t_i in $I^2_{\pi_i(j)}$. Observe that if I is a yes-instance then we can still serve all the requests in the new instance. **However, when I is a no-instance, then there is a family $\mathcal{F}$ of $2k$ many k-tuples from $[k]^2$ so that each tuple represents k requests that cannot be served simultaneously.** Each tuple corresponds to some k requests that have to be routed through a single copy of I, which is impossible when I is a no-instance.

We can now iterate this argument. In the next step we repeat this construction k times (but possibly with different permutations), place such k instances next to each other, and create the third layer comprising now k^2 copies of I. Then for each $i \in [k]$ we need a permutation $\pi_i \colon [k^2] \to [k^2]$ describing the connections between the sinks from the second layer to the sources from the third layer. Again, if I is a no-instance, we obtain a family $\mathcal{F}$ of $3k^2$ many k-tuples from $[k]^3$ corresponding to subsets of requests that cannot be served simultaneously. We want to show that after $d = f(k)$ many iterations no subset A of 50% requests can be served. In other words, the family $\mathcal{F}$ should always contain a tuple contained in A, certifying that A is not realizable. This will give a reduction from the exact version of Dag Disjoint Paths to a version of Dag Disjoint Paths with gap $\frac{1}{2}$. The crux of the proof is to find a collection of permutations that will guarantee the desired property of $\mathcal{F}$.

It is convenient to think about this construction as a game in which the first player chooses the permutations governing the connections between the layers (thus creating an instance of Dag Disjoint Paths) and the second player picks a subset A of 50% requests. The first player wins whenever the family $\mathcal{F}$ of forbidden k-tuples includes a tuple contained in A. We need to show that the first player has a single winning strategy against every possible strategy of the second player. We will prove that a good strategy for the first player is to choose every permutation independently and uniformly at random. In fact, for a sufficiently large d and any fixed strategy A of the second player, the probability that A wins against a randomized strategy is smaller than 2^{-k^d}. Since the number of possible strategies for the second player is at most 2^{k^d} (because there are k^d requests), **the union bound implies that the first player has a positive probability of choosing a strategy that guarantees a victory against every strategy of the second player.** This translates to the existence of a family of permutations for which the gap amplification works.

2 Preliminaries

We follow the convention $[n] = \{1, 2, \ldots, n\}$ and use the standard graph theoretic terminology from Diestel's book [18]. We begin by formalizing the problem.

MAX DISJOINT PATHS **Parameter:** k
Input: A digraph D, a set $\mathcal{T}$ of k pairs $(s_i, t_i) \in V(D)^2$.
Task: Find a largest collection $\mathcal{P}$ of vertex-disjoint paths so that each path $P \in \mathcal{P}$ is an (s_i, t_i)-path for some $(s_i, t_i) \in \mathcal{T}$.

We refer to the pairs from $\mathcal{T}$ as *requests*. A solution $\mathcal{P}$ is said to *serve* request (s_i, t_i) if it contains an (s_i, t_i)-path. The condition of vertex-disjointedness implies that each request can be served by at most one path in $\mathcal{P}$. A *yes-instance* is an instance admitting a solution serving all the k-requests. Otherwise we deal with a *no-instance*. (MAX) DAG DISJOINT PATHS is a variant of (MAX) DISJOINT PATHS where the input digraph is assumed to be acyclic.

Notation for trees. For a rooted tree T and $v \in V(T)$ we denote by $\mathsf{Children}(v)$ the set of direct descendants of v. A vertex v in a rooted tree is a leaf if $\mathsf{Children}(v) = \emptyset$. We refer to the set of leaves of T as $L(T)$. The depth of a vertex $v \in V(T)$ is defined as its distance from the root, measured by the number of edges. In particular, the depth of the root equals 0. The set of vertices of depth i in T is called the i-th layer of T.

For $v \in V(T)$ we write T^v to denote the subtree of T rooted at v. We can additionally specify an integer $\ell \geq 1$ and write $T^{v,\ell}$ for the tree comprising the first ℓ layers of T^v. In particular, the tree $T^{v,1}$ contains only the vertex v.

For $k, d \in \mathbb{N}$ we denote by $T_{k,d}$ the full k-ary rooted tree of depth d. We have $|L(T_{k,d})| = k^d$. A subset $A \subseteq L(T_{k,d})$ is called a q-subset for $q \in \mathbb{N}$ if $|A| \geq |L(T_{k,d})| \,/\, q$.

Fixed parameter tractability. We provide only the necessary definitions here; more information can be found in the book [16]. A parameterized problem can be formalized as a subset of $\Sigma^* \times \mathbb{N}$. We say that a problem is *fixed parameter tractable* (FPT) if it admits an algorithm solving an instance (I, k) in running time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is some computable function.

To argue that a parameterized problem is unlikely to be FPT, we employ FPT-reductions that run in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ and transform an instance (I, k) into an equivalent one (I', k') where $k' = g(k)$. A canonical parameterized problem that is believed to lie outside the class FPT is k-CLIQUE. The problems that are FPT-reducible to k-CLIQUE form the class W[1].

Negative association. We introduce the following concept necessary for our probabilistic argument. There are several definitions capturing negative dependence between random variables; intuitively it means that when one variable takes a high value then a second one is more likely to take a low value. Negative association formalizes this idea in a strong sense.

▶ **Definition 2.** *A collection of random variables $X_1, X_2, \ldots, X_n \in \mathbb{R}$ is said to be* negatively associated *if for every pair of disjoint subsets $A_1, A_2 \subseteq [n]$ and every pair of increasing functions $f_1 \colon \mathbb{R}^{|A_1|} \to \mathbb{R}$, $f_2 \colon \mathbb{R}^{|A_2|} \to \mathbb{R}$ it holds that*

$$\mathbb{E}\left[f_1(X_i \mid i \in A_1) \cdot f_2(X_i \mid i \in A_2)\right] \le \mathbb{E}\left[f_1(X_i \mid i \in A_1)\right] \cdot \mathbb{E}\left[f_2(X_i \mid i \in A_2)\right].$$

We make note of several important properties of negative association.

▶ **Lemma 3** ([28, Prop. 3, 6, 7])**.** *Consider a collection of random variables $X_1, X_2, \ldots, X_n \in \mathbb{R}$ that is negatively associated. Then the following properties hold.*

1. *For every family of disjoint subsets $A_1, \ldots, A_k \subseteq [n]$ and increasing functions $f_1, \ldots, f_k$, $f_i \colon \mathbb{R}^{|A_i|} \to \mathbb{R}$, the collection of random variables*

$$f_1(X_i \mid i \in A_1),\ f_2(X_i \mid i \in A_2),\ \ldots,\ f_k(X_i \mid i \in A_k)$$

   is negatively associated.
2. *If random variables $Y_1, \ldots, Y_n$ are negatively associated and independent from $X_1, \ldots, X_n$ then the collection $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ is negatively associated.*
3. *For every sequence $(x_1, x_2, \ldots, x_n)$ of real numbers we have*

$$\mathbb{P}\left[X_i \le x_i \mid i \in [n]\right] \le \prod_{i=1}^{n} \mathbb{P}\left[X_i \le x_i\right].$$

▶ **Lemma 4.** *Let $n, k \in \mathbb{N}$. For $i \in [k]$ let $\mathcal{X}^i = (X^i_1, \ldots, X^i_n)$ be a sequence of real random variables that are negatively associated. Suppose that $\mathcal{X}^1, \ldots, \mathcal{X}^k$ are independent from each other. Then the random variables $(\sum_{i=1}^k X^i_1,\ \ldots,\ \sum_{i=1}^k X^i_n)$ are negatively associated.*

Proof. By Lemma 3(2) we know that the union $\mathcal{X}^1 \cup \cdots \cup \mathcal{X}^k$ forms a collection of nk random variables that are negatively associated. We divide it into n disjoint subsets of the form $(\{X^1_j, \ldots, X^k_j\})_{j=1}^n$ and apply Lemma 3(1) for the increasing function $f \colon \mathbb{R}^k \to \mathbb{R}$, $f(x_1, \ldots, x_k) = \sum_{i=1}^k x_i$. ◀

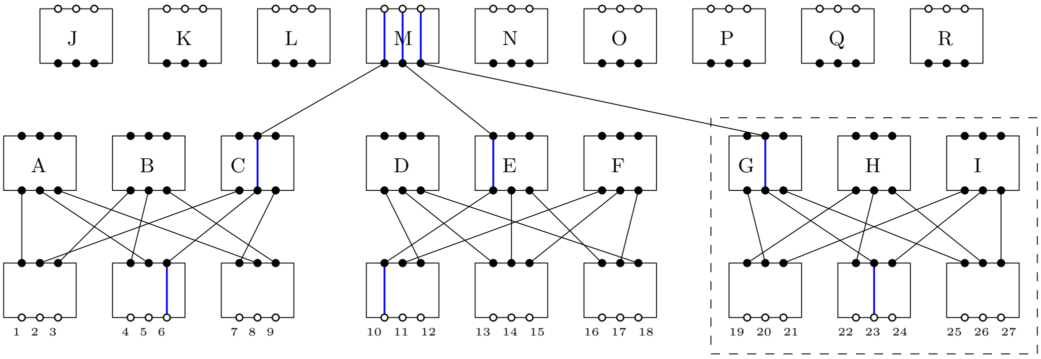
Negative association occurs naturally in situations like random sampling without replacement. A scenario important for us is when an ordered sequence of numbers is being randomly permuted. Intuitively, observing a high value at some index removes this value from the pool and decreases the chances of seeing high values at the remaining indices.

▶ **Theorem 5** ([28, Thm. 2.11])**.** *Consider a sequence $(x_1, x_2, \ldots, x_n)$ of real numbers. Let $\Pi \colon [n] \to [n]$ be a random variable representing a permutation of the set $[n]$ chosen uniformly at random. For $i \in [n]$ we define a random variable $X_i = x_{\Pi^{-1}(i)}$. Then the random variables $X_1, X_2, \ldots, X_n$ are negatively associated.*

3 The reduction

Our main objects of interest are collections of functions associated with the nodes of the full k-ary rooted tree. Such a function for a node v gives an ordering of leaves in the subtree of v.

▶ **Definition 6.** *A* scheme *for $T_{k,d}$ is a collection of functions, one for each node in $T_{k,d}$, such that the function f_v associated with $v \in V(T_{k,d})$ is a bijection from $L(T^v_{k,d})$ to $[|L(T^v_{k,d})|]$. Let* Schemes(k, d) *denote the family of all schemes for $T_{k,d}$.*



J	K	L	N	O	M	P	R	Q	M	Q	L	N	O	J	P	R	K	P	K	L	N	M	O	J	R	Q
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

A	C	B	B	A	C	C	A	B	E	F	D	D	E	F	E	F	D	H	G	I	H	G	I	G	H	I
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Figure 1 An illustration for Definition 7 with $k = d = 3$. The boxes represent copies of an instance I with $|\mathcal{T}| = 3$, the large instance is $J_{3,3}(I, \beta)$ for the scheme β listed at the bottom, and the dashed rectangle surrounds the instance $J_3 = J_{3,2}(I, \beta_3)$ where β_3 is a truncation of β to the right subtree of $T_{3,3}$. The hollow disks represent the sinks and sources on the large instance. All the arcs are oriented upwards. The leaves of $T_{3,3}$ are numbered as $1, 2, \ldots, 27$. For the sake of legibility, most of the arcs in the last layer are omitted and the copies of the original instance within layers $2, 3$ are marked with letters. The letters are also used in the representation of the scheme β which contains 9 bijections between sets of size 3 and 3 bijections between sets of size 9 (and one bijection for size 27, which is immaterial here). The blue lines exemplify vertex pairs which belong to the request set of the large instance; the sources (in the layer 1) indexed by $6, 10, 23$ are mapped to the sinks in the copy M (in the layer 3). If a subset $\Gamma \subseteq [27]$ includes $6, 10, 23$ then it has a collision with respect to the scheme β. If we work with a no-instance then such a subset Γ of requests cannot be served as this would require routing three of them through the copy M.

We will now formalize the idea of connecting multiple copies of an instance. On an intuitive level, we construct a d-layered instance by taking k many $(d-1)$-layered instances and adding a new layer comprising k^{d-1} copies of the original instance I. Then we map the sinks in the layer $(d-1)$ to the sources in the layer d according to k bijections read from a scheme. These mappings govern how we place the arcs towards the layer d and which vertex pairs form the new request set. We need a scheme $\beta \in \mathsf{Schemes}(k, d)$ to arrange all the arcs between the layers.

In order to simplify the notation we introduce the following convention. Suppose that an instance J is being build with multiple disjoint copies of an instance $I = (D, k, \mathcal{T})$, referred to as $I_1, I_2, \ldots$. Then we refer to the copy of the vertex $s_i \in V(D)$ (resp. t_i) in I_j as $I_j[s_i]$ (resp. $I_j[t_i]$).

▶ **Definition 7.** *Given an instance $I = (D, k, \mathcal{T})$ of* Dag Disjoint Paths *and a scheme $\beta = (f_v)_{v \in V(T_{k,d})} \in \mathsf{Schemes}(k, d)$ we construct an instance $J_{k,d}(I, \beta) = (D', k^d, \mathcal{T}')$ of* Dag Disjoint Paths*. The elements of $\mathcal{T}'$ will be indexed by the leaves of $T_{k,d}$ as $(s_v, t_v)_{v \in L(T_{k,d})}$ while the elements of $\mathcal{T}$ (in the instance I) are indexed by $1, \ldots, k$ as $(s_i, t_i)_{i \in [k]}$.*

If $d = 1$, we simply set $J_{k,1}(I, \beta) = I$, ignoring β. We index $\mathcal{T}$ by $L(T_{k,1})$ in an arbitrary order.

Consider $d > 1$. Let r be the root of $T_{k,d}$ with $\mathsf{Children}(r) = \{u_1, \ldots, u_k\}$. For $i \in [k]$ let β_i be the truncation of β to the nodes in the subtree $T^{u_i}_{k,d}$ and $J_i = (D_i, k^{d-1}, \mathcal{T}_i)$ be the instance $J_{k,d-1}(I, \beta_i)$. We take a disjoint union of $J_1, \ldots, J_k$ and k^{d-1} copies of I referred to as $I_1, I_2, \ldots$ (see Figure 1). These k^{d-1} copies of I form layer d.

Recall that for $i \in [k]$ the bijection f_{u_i} maps $L(T^{u_i}_{k,d})$ to $[k^{d-1}]$. For each $i \in [k]$ and $v \in L(T^{u_i}_{k,d})$ we insert an arc from $J_i[t_v]$ to $I_{f_{u_i}(v)}[s_i]$. Then we add the pair $(J_i[s_v], I_{f_{u_i}(v)}[t_i])$ to $\mathcal{T}'$. This pair is assigned index $\iota(v)$ in $\mathcal{T}'$ where ι is the natural embedding $L(T^{u_i}_{k,d}) \to L(T_{k,d})$.

Note that whenever D is acyclic then D' is acyclic as well so the procedure indeed outputs an instance of Dag Disjoint Paths. It is also clear that when I admits a solution serving all the k requests, it can be used to serve all the requests in $J_{k,d}(I, \beta)$.

▶ **Observation 8.** *Let $k, d \in \mathbb{N}$ and $\beta \in \mathsf{Schemes}(k, d)$. If $I = (D, k, \mathcal{T})$ is a yes-instance of Dag Disjoint Paths then $J_{k,d}(I, \beta)$ is a yes-instance as well.*

The case when I is a no-instance requires a more careful analysis. We introduce the notion of a *collision* that certifies that some subset of requests cannot be served.

▶ **Definition 9.** *Let $k, d \in \mathbb{N}$, $A \subseteq L(T_{k,d})$, and $\beta = (f_v)_{v \in V(T_{k,d})} \in \mathsf{Schemes}(k, d)$. We say that $u \in V(T_{k,d})$ forms a* collision *with respect to (A, β) if A contains elements $a_1, \dots, a_k$ such that:*

1. *for each $i \in [k]$ the node a_i is a descendant of $u_i \in \mathsf{Children}(u)$ where $u_1, \dots, u_k$ are distinct,*
2. *$f_{u_1}(a_1) = f_{u_2}(a_2) = \cdots = f_{u_k}(a_k)$.*

▶ **Lemma 10.** *Let $k, d \in \mathbb{N}$, $A \subseteq L(T_{k,d})$, and $\beta = (f_v)_{v \in V(T_{k,d})} \in \mathsf{Schemes}(k, d)$. Suppose that there exists a collision with respect to (A, β). Let $I = (D, k, \mathcal{T})$ be a no-instance of Dag Disjoint Paths. Then no solution to the instance $(D', k^d, \mathcal{T}') = J_{k,d}(I, \beta)$ can simultaneously serve all the requests $\{(s_v, t_v)_{v \in A}\}$.*

Proof. We will prove the lemma by induction on d. In the case $d = 1$ we have $J_{k,1}(I, \beta) = I$ and the only possibility of a collision is when $A = L(T_{k,1})$ so $\{(s_v, t_v)_{v \in A}\}$ is the set of all the requests. By definition, we cannot serve all the requests in a no-instance. Let us assume $d > 1$ from now on.

First suppose that the collision occurs at the root $r \in V(T_{k,d})$. Let $\mathsf{Children}(r) = \{u_1, \dots, u_k\}$. Then there exists $A' = \{a_1, \dots, a_k\} \subseteq A$ such that a_i is a descendant of u_i and $f_{u_1}(a_1) = f_{u_2}(a_2) = \cdots = f_{u_k}(a_k)$. We refer to this common value as $x = f_{u_i}(a_i)$. We will also utilize the notation from Definition 7.

Observe that in order to serve the request (s_{a_i}, t_{a_i}) in D' the path P_i starting at $s_{a_i} = J_i[s_{a_i}]$ must traverse the arc from $J_i[t_{a_i}]$ to $I_x[s_i]$ as every other arc leaving D_i leads to some I_y with $y \neq x$ having no connection to $t_{a_i} = I_x[t_i]$. Furthermore, the path P_i must contain a subpath connecting $I_x[s_i]$ to $I_x[t_i]$ in I_x. Since the same argument applies to every $i \in [k]$, we would have to serve all the k requests in I_x. But this is impossible because I_x is a copy of I which is a no-instance.

Now suppose that the collision does not occur at the root. Then it must occur in the subtree $T^{u_i}_{k,d}$ for some $i \in [k]$. For every $v \in A$ being a descendant of u_i, any path P_v serving the request (s_v, t_v) in D' must contain a subpath P'_v in D_i from $J_i[s_v]$ to $J_i[t_v]$ as again it must leave D_i through the vertex $J_i[t_v]$. By the inductive assumption, we know that we cannot simultaneously serve all the requests $(s_v, t_v)_{v \in A \cap L(T^{u_i}_{k,d})}$ in the smaller instance J_i. The lemma follows. ◀

We can now state our main technical theorem. Recall that a subset $A \subseteq L(T_{k,d})$ is called a q-subset if $|A| \geq |L(T_{k,d})|/q = k^d/q$.

▶ **Theorem 11.** *Let $k, d, q \in \mathbb{N}$ satisfy $d \geq k \cdot (4q)^{4k \log k}$. Then there exists $\beta \in \mathsf{Schemes}(k, d)$ such that for every q-subset $A \subseteq L(T_{k,d})$ there is a collision with respect to (A, β).*

The proof is postponed to Section 4 which abstracts from the DISJOINT PATHS problem and focuses on random permutations. With Theorem 11 at hand, the proof of the main result is easy.

▶ **Theorem 1.** *Let $q \in \mathbb{N}$ be a constant. It is W[1]-hard to distinguish whether for a given instance of k-DAG DISJOINT PATHS:*

1. *all the requests can be served simultaneously, or*
2. *no set of k/q requests can be served simultaneously.*

Proof. We are going to give an FPT-reduction from the exact variant of k-DAG DISJOINT PATHS, which is W[1]-hard [51], to the variant with a sufficiently large gap. To this end, we present an algorithm that, given an instance $I = (D, k, \mathcal{T})$, runs in time $f(k, q) \cdot |I|$ and outputs an instance $J = (D', k', \mathcal{T}')$ such that:

1. k' depends only on k and q,
2. if I is a yes-instance then J is a yes-instance, and
3. if I is a no-instance then no solution to J can simultaneously serve at least k'/q requests.

Obviously, being able to separate these two cases for J (all requests vs. at most $\frac{1}{q}$-fraction of requests) is sufficient to determine whether I is a yes-instance.

We set $d = k \cdot (4q)^{4k\log(k)}$ accordingly to Theorem 11. It guarantees that there exists a scheme $\beta \in \mathsf{Schemes}(k, d)$ such that for every q-subset $A \subseteq L(T_{k,d})$ there is a collision with respect to (A, β). Observe that such a scheme can be computed in time $f(k, q)$ because d is a function of (k, q) and the size of the family $\mathsf{Schemes}(k, d)$ is a function of (k, d). The same holds for the number of all q-subsets $A \subseteq L(T_{k,d})$. Therefore, we can simply iterate over all $\beta \in \mathsf{Schemes}(k, d)$ and check for each q-subset A whether there is a collision or not.

The instance J is defined as $J_{k,d}(I, \beta)$. A direct implementation of Definition 7 takes time $f(k, d) \cdot |I|$. Observation 8 says that if I is a yes-instance, then J is as well, whereas Lemma 10 ensures that if I is a no-instance, then for each set of k'/q requests (corresponding to some q-subset $A \subseteq L(T_{k,d})$ which must have a collision with β) no solution can simultaneously serve all of them. This concludes the correctness proof of the reduction. ◀

We remark that Theorem 1 works in a more general setting, where q is not necessarily a constant, but a function of k. This enables us to rule out not only an $\mathcal{O}(1)$-approximation in FPT time, but also an $\alpha(k)$-approximation for some slowly growing function $\alpha(k) \to \infty$. However, the value of the parameter k' becomes k^d for $d = \Omega(q^{k\log k})$ so q ends up very small compared to the new parameter k'. This is only sufficient to rule out approximation factors of the form $\alpha(k) = (\log k)^{o(1)}$. A detailed analysis of how to adjust such parameters is performed in [53].

4 Constructing the scheme

This section is devoted to the proof of Theorem 11. Before delving into the rigorous analysis, we sketch the main ideas behind the proof.

Outline. We use the probabilistic method to prove the existence of a scheme having a collision with every q-subset of leaves in $T_{k,d}$. We will show that for a sufficiently large d choosing each bijection at random yields a very high probability of a collision with any fixed q-subset. Specifically, the probability that a collision does not occur should be less than 2^{-k^d}. Since the number of all q-subsets of a k^d-size set is bounded by 2^{k^d}, the union bound will imply that the probability that a collision does not occur for at least one q-subset is strictly less than one, implying the existence of the desired scheme.

Let us fix a q-subset $A \subseteq L(T_{k,d})$. Suppose there is a vertex $u \in V(T_{k,d})$ such that for every child y of u the fraction of leaves in $T^y_{k,d}$ belonging to A is at least $1/q$. Let ℓ denote $[|L(T^y_{k,d})|]$. For each such child we choose a random bijection from $L(T^y_{k,d})$ to $[\ell]$. The probability that each of these k bijections maps an element of A to a fixed index $x \in [\ell]$ is at least q^{-k}. Such events are not independent for distinct x but we will see that they are negatively associated, which still allows us to upper bound the probability of no such event happening by $(1-q^{-k})^\ell$ (see Lemma 12).

How to identify such a vertex u? First, it is sufficient for us to relax the bound $1/q$ assumed above to $1/(4q)$. Observe that for each layer in $T_{k,d}$ there must be many vertices v satisfying $|A \cap L(T^v_{k,d})| \geq \frac{1}{2q}|L(T^v_{k,d})|$. Suppose that v does not meet our criterion: this means that it has a child v' with less than $1/(4q)$-fraction of the A-leaves in its subtree. But then the average fraction of the A-leaves among the remaining children is higher than the fraction for v. Consequently, we can choose a child of v with a higher fraction and repeat this argument inductively. We show that after $\mathcal{O}(k\log(q))$ many steps this process must terminate so we are guaranteed to find a vertex for which every child has at least a $1/(4q)$-fraction of the A-leaves. This is proven in Lemma 13.

Finally, to obtain a large probability of a collision we must show that there many such vertices u with a large sum of their subtrees' sizes. This will allows us to multiply the aforementioned bounds of the form $(1-q^{-k})^\ell$ with a large sum of the exponents ℓ. By applying the argument above to a single layer in $T_{k,d}$ we can find such a collection with the sum of their subtrees' sizes being k^d divided by some function of k and q. But we can also apply it to multiple layers as long as they are sufficiently far from each other (so that the vertices found by the inductive procedure are all distinct). Therefore, it suffices to take d large enough so that the number of available layers surpasses the factors in the denominator, which depend only on k and q. This is analyzed in Lemma 15.

We begin with a probabilistic lemma stating that randomly permuting k large subsets of a common universe yields a large chance of creating a non-empty intersection of these sets.

▶ **Lemma 12.** *Let $k, z, \ell \in \mathbb{N}$ and $X_1, \dots, X_k$ be subsets of $[\ell]$ of size at least ℓ/z each. Next, let $\Pi_1, \dots, \Pi_k \colon [\ell] \to [\ell]$ be independent random variables with a uniform distribution on the family of all permutations over the set $[\ell]$. Then*

$$\mathbb{P}\left[\Pi_1(X_1) \cap \Pi_2(X_2) \cap \dots \cap \Pi_k(X_k) = \emptyset\right] \leq \exp(-\ell/z^k).$$

Proof. For $i \in [k]$ and $j \in [\ell]$ let $Y^i_j = 1$ if $j \in \Pi_i(X_i)$ and $Y^i_j = 0$ otherwise. By Theorem 5 the variables $(Y^i_1, \dots, Y^i_\ell)$ have negative association for each $i \in [k]$. Note that $\mathbb{E}Y^i_j \geq 1/z$. Next, let $Z_j = \sum_{i=1}^k Y^i_j$ for $j \in [\ell]$. Lemma 4 ensures that the variables $Z_1, \dots, Z_\ell$ also enjoy negative association. Condition $\Pi_1(X_1) \cap \Pi_2(X_2) \cap \dots \cap \Pi_k(X_k) = \emptyset$ is equivalent to $\max(Z_j)_{j=1}^\ell \leq k-1$. We have

$$\mathbb{P}\left[Z_j \leq k-1\right] = 1 - \mathbb{P}\left[Z_j = k\right] = 1 - \prod_{i=1}^k \mathbb{P}\left[j \in \Pi_i(X_i)\right] \leq 1 - 1/z^k.$$

$$\mathbb{P}\left[\max(Z_j)_{j=1}^\ell \leq k-1\right] \leq \prod_{j=1}^\ell \mathbb{P}\left[Z_j \leq k-1\right] \leq (1-1/z^k)^\ell = (1-1/z^k)^{z^k \cdot (\ell/z^k)} \leq \exp(-\ell/z^k).$$

In the first inequality we used Lemma 3(3). The last one holds because $(1-\frac{1}{m})^m < \frac{1}{e}$ for all $m \geq 2$. ◀

Notation. We introduce some additional notation to work with the tree $T_{k,d}$. For a vertex $v \in V(T_{k,d})$ let $\mathsf{Leaves}(v)$ denote the size of the set $L(T^v_{k,d})$. Note that $\mathsf{Leaves}(v) = k^{d-h}$ where h is the depth of v. Next, for a set $A \subseteq L(T_{k,d})$, we will write $\mathsf{Frac}_A(v) = |A \cap L(T^v_{k,d})| \,/\, \mathsf{Leaves}(v)$. When A is clear from the context, we will omit the subscript.

▶ **Lemma 13.** *Let $k, d, q, \tau \in \mathbb{N}$ satisfy $k, q \geq 2$ and $d \geq \tau \geq 2k \cdot \log(q)$. Next, let $v \in V(T_{k,d})$ be of depth at most $d - \tau$ and $A \subseteq L(T_{k,d})$ satisfy $\mathsf{Frac}_A(v) \geq \frac{1}{q}$. Then there exists a vertex $u \in V(T^{v,\tau}_{k,d})$ such that for each $y \in \mathsf{Children}(u)$ it holds that $\mathsf{Frac}_A(y) \geq \frac{1}{2q}$.*

Proof. Suppose the claim does not hold. We will show that under this assumption for each $i \in [\tau]$ there exists $v_i \in V(T^{v,i}_{k,d})$ with $\mathsf{Frac}(v_i) \geq \frac{1}{q} \cdot (1 + \frac{1}{2k-2})^{i-1}$. Then by substituting $i = \tau > (2k-2) \cdot \log(q)$ and estimating $(1 + \frac{1}{m})^m > 2$ (for all $m \geq 2$) we will arrive at a contradiction:

$$\mathsf{Frac}(v_\tau) \geq \frac{1}{q} \cdot \left(1 + \frac{1}{2k-2}\right)^{(2k-2)\cdot\log(q)} > \frac{1}{q} \cdot 2^{\log(q)} \geq 1.$$

We now construct the promised sequence (v_i) inductively. For $i = 1$ we set $v_1 = v$ which obviously belongs to $T^{v,1}_{k,d}$ and satisfies $\mathsf{Frac}(v) \geq \frac{1}{q}$. To identify v_{i+1} we consider $\mathsf{Children}(v_i) = u_1, u_2, \ldots, u_k$. We have $\mathsf{Frac}(v_i) = \frac{1}{k} \cdot \sum_{j=1}^{k} \mathsf{Frac}(u_j)$. We define v_{i+1} as the child of v_i that maximizes the value of Frac (see Figure 2). By the assumption, one of the children satisfies $\mathsf{Frac}(u_j) < \frac{1}{2q}$. Then $\mathsf{Frac}(v_{i+1})$ is lower bounded by the average value of Frac among the remaining $k-1$ children, which is at least $\frac{1}{k-1}\left(\mathsf{Frac}(v_i) \cdot k - \frac{1}{2q}\right)$. We have $\mathsf{Frac}(v_i) \geq \mathsf{Frac}(v_1) \geq \frac{1}{q}$ so $\left(\mathsf{Frac}(v_i) \cdot k - \frac{1}{2q}\right) \geq \left(\mathsf{Frac}(v_i) \cdot k - \frac{\mathsf{Frac}(v_i)}{2}\right)$. We check that v_{i+1} meets the specification:

$$\mathsf{Frac}(v_{i+1}) \geq \frac{\mathsf{Frac}(v_i)}{k-1} \cdot \left(k - \frac{1}{2}\right) = \mathsf{Frac}(v_i) \cdot \left(1 + \frac{1}{2k-2}\right) \geq \frac{1}{q} \cdot \left(1 + \frac{1}{2k-2}\right)^i$$

In the last inequality we have plugged in the inductive assumption. The lemma follows. ◀

To apply Lemma 13 we need to identify many vertices satisfying $\mathsf{Frac}_A(v) \geq \frac{1}{2q}$. To this end, we will utilize the following simple fact.

▶ **Lemma 14.** *Let $a_1, a_2, \ldots, a_\ell \in [0,1]$ be a sequence with mean at least x for some $x \in [0,1]$. Then at least $\frac{x\ell}{2}$ elements in the sequence are lower bounded by $\frac{x}{2}$.*

Proof. Suppose that $|\{a_i \geq \frac{x}{2} \mid i \in [\ell]\}| < \frac{x\ell}{2}$. This leads to a contradiction:

$$\sum_{i=1}^{\ell} a_i < 1 \cdot \frac{x\ell}{2} + \frac{x}{2} \cdot \ell = x\ell. \qquad ◀$$

We will use the lemmas above for a fixed layer in the tree $T_{k,d}$ to identify multiple vertices v meeting the requirements of Lemma 13. For each such v we can find a close descendant u of v for which we are likely to observe a collision. The value ℓ in Lemma 12, governing the probability of a collision, corresponds to the number of leaves in the subtree of u, i.e., $\mathsf{Leaves}(u)$. Since this value appears in the exponent of the formula, we need a collection of such vertices u in which the total sum of $\mathsf{Leaves}(u)$ is large.

▶ **Lemma 15.** *Let $k, d, q \in \mathbb{N}$ satisfy $k, q \geq 2$, $d \geq 4kq$. If $A \subseteq L(T_{k,d})$ is a q-subset then there exists a set $F \subseteq V(T_{k,d})$ with the following properties.*

1. *For each $v \in F$ and $u \in \mathsf{Children}(v)$ it holds that $\mathsf{Frac}_A(u) \geq \frac{1}{4q}$.*
2. *The sum $\sum_{v \in F} \mathsf{Leaves}(v)$ equals at least $d \cdot k^d \cdot (4q)^{-3k\log(k)}$.*

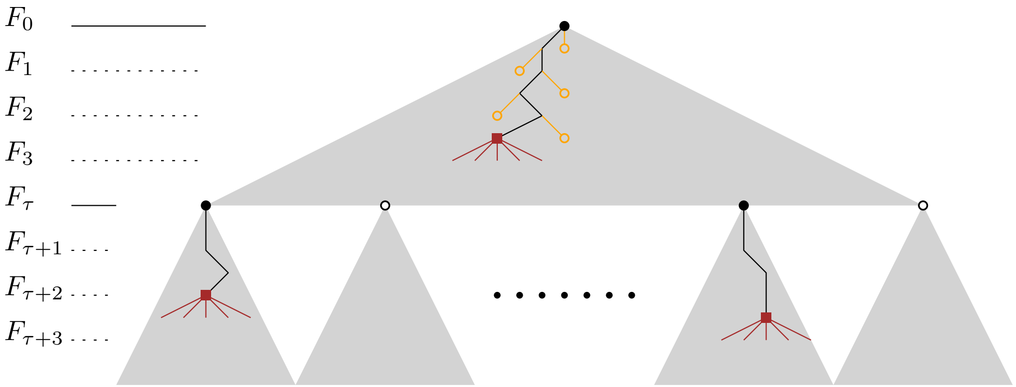


Figure 2 An illustration for Lemma 15. We consider layers $F_0, F_\tau, F_{2\tau}, \ldots$ The vertices from F_0^+ and F_τ^+ are marked by black disks and their subtrees $F_{k,d}^{v,\tau}$ are depicted as gray triangles. For each vertex $v \in F^+$ we apply Lemma 13 to identify a vertex $\gamma(v) \in F$: the red square inside the corresponding triangle. The root also illustrates the argument from Lemma 13. We start with a vertex v satisfying $\mathsf{Frac}(v) \ge \frac{1}{2q}$ and while one of its children v' has $\mathsf{Frac}(v') < \frac{1}{4q}$ we can find another child v'' of v with $\mathsf{Frac}(v'') > \mathsf{Frac}(v)$. This process terminates within τ steps.

Proof. Let $F_i \subseteq V(T_{k,d})$ be i-th layer of $T_{k,d}$, i.e., the set of vertices of depth i; we have $|F_i| = k^i$ and $\mathsf{Leaves}(v) = k^{d-i}$ for each $v \in F_i$. Since their subtrees are disjoint, we can see that $\sum_{v\in F_i} |A \cap L(T_{k,d}^v)| = |A|$. Therefore $\sum_{v\in F_i} \mathsf{Frac}(v)/|F_i| \ge \frac{1}{q}$. By Lemma 14 at least $\frac{1}{2q}$ fraction of the vertices in F_i must satisfy $\mathsf{Frac}(v) \ge \frac{1}{2q}$. Let us denote this subset as F_i^+.

Let $\tau = \lceil 2k \cdot \log(2q) \rceil$ and $M = \lfloor d/\tau \rfloor$. We define $F^+ = F_0^+ \cup F_\tau^+ \cup F_{2\tau}^+ \cup \cdots \cup F_{(M-1)\tau}^+$. Observe that for each pair $u, v \in F^+$ the trees $T_{k,d}^{u,\tau}, T_{k,d}^{v,\tau}$ are disjoint. We apply Lemma 13 with $q' = 2q$ to each $v \in F^+$ to obtain a vertex $\gamma(v) \in V(T_{k,d}^{v,\tau})$ satisfying condition (1). The disjointedness of these subtrees ensures that the vertices $\gamma(v)_{v\in F^+}$ are distinct. We define $F = \{\gamma(v) \mid v \in F^+\}$.

Now we take care of condition (2). Let us fix $j \in [0, M-1]$. Since $\gamma(v) \in V(T_{k,d}^{v,\tau})$ for v with the depth $j\tau$, we infer that the depth of $\gamma(v)$ is at most $(j+1)\tau - 1$ so $|\mathsf{Leaves}(\gamma(v))| \ge k^{d+1-(j+1)\tau}$. We have established already that $|F_{j\tau}^+| \ge \frac{|F_{j\tau}|}{2q} = \frac{k^{j\tau}}{2q}$. The assumption $d \ge 4kq$ implies $d \ge \tau$ so we can simplify $M = \lfloor d/\tau \rfloor \ge d/(2\tau)$. We estimate the sum within each layer $F_{j\tau}^+$ and then multiply it by M.

$$\sum_{v\in F_{j\tau}^+} \mathsf{Leaves}(\gamma(v)) \ge \frac{k^{j\tau}}{2q} \cdot k^{d+1-(j+1)\tau} = \frac{k^{d+1-\tau}}{2q}$$

$$\sum_{v\in F} \mathsf{Leaves}(v) = \sum_{j=0}^{M-1} \sum_{v\in F_{j\tau}^+} \mathsf{Leaves}(\gamma(v)) \ge \frac{d \cdot k^{d+1-\tau}}{2\tau \cdot 2q}$$

To get rid of the ceiling, we estimate $\tau \le 2k \cdot \log(4q)$. Then $k^\tau \le k^{2k\log(4q)} = (4q)^{2k\log(k)}$. We also use a trivial bound $\tau \le 4kq$. We can summarize the analysis by

$$\sum_{v\in F} \mathsf{Leaves}(v) \ge \frac{d \cdot k^{d+1-\tau}}{2\tau \cdot 2q} = \frac{d \cdot k^{d+1}}{k^\tau \cdot 4q\tau} \ge \frac{d \cdot k^{d+1}}{(4q)^{2k\log(k)} \cdot 16kq^2} \ge \frac{d \cdot k^d}{(4q)^{3k\log(k)}} \qquad ◀$$

Now we combine the gathered ingredients to show that a random scheme yields a high probability of a collision with any fixed q-subset. At this point we also adjust d to be larger then the factors depending on k and q.

▶ **Lemma 16.** *Let $k, d, q \in \mathbb{N}$ satisfy $d \geq k \cdot (4q)^{4k \log k}$. Consider some q-subset $A \subseteq L(T_{k,d})$. Suppose that we choose the scheme $\beta = (f_v)_{v \in V(T_{k,d})} \in \mathsf{Schemes}(k, d)$ by picking each bijection $f_v \colon L(T^v_{k,d}) \to [|L(T^v_{k,d})|]$ uniformly and independently at random. Then the probability that (A, β) has no collision is at most $\exp(-k^d)$.*

Proof. We apply Lemma 15 and use the obtained set $F \subseteq V(T_{k,d})$ to analyze the probability of getting a collision. Consider $u \in F$ with $\mathsf{Children}(u) = \{u_1, \dots, u_k\}$ and let C_u denote the event that (A, β) has a collision at u. For each $i \in [k]$ we have $\mathsf{Leaves}(u_i) = \mathsf{Leaves}(u)/k$ and we know from Lemma 15(1) that $\mathsf{Frac}_A(u_i) \geq 1/(4q)$. For each $i \in [k]$ a random bijection f_{u_i} is chosen between $L(T^{u_i}_{k,d})$ and $[\mathsf{Leaves}(u_i)]$. This can be interpreted as first picking an arbitrary bijection to $[\mathsf{Leaves}(u_i)]$ and then combining it with a random permutation over $[\mathsf{Leaves}(u_i)]$. We apply Lemma 12 with $z = 4q$ to infer that the probability of getting no collision at u is upper bounded by

$$\mathbb{P}[\neg C_u] \leq \exp\left(\frac{-\mathsf{Leaves}(u_i)}{z^k}\right) = \exp\left(\frac{-\mathsf{Leaves}(u)}{k \cdot (4q)^k}\right).$$

Since the sets $(\mathsf{Children}(u))_{u \in F}$ are pairwise disjoint, the corresponding events C_u are independent. We can thus upper bound the probability of getting no collision at all by the product $\prod_{u \in F} \mathbb{P}[\neg C_u]$. Next, by Lemma 15(2) and the assumption on d we know that

$$\sum_{u \in F} \mathsf{Leaves}(u) \geq d \cdot k^d \cdot (4q)^{-3k \log k} \geq k^{d+1} \cdot (4q)^k.$$

We combine this with the previous formula to obtain

$$\mathbb{P}\left[\neg \bigcup_{u \in F} C_u\right] = \mathbb{P}\left[\bigcap_{u \in F} \neg C_u\right] = \prod_{u \in F} \mathbb{P}[\neg C_u] \leq \exp\left(\frac{-\sum_{u \in F} \mathsf{Leaves}(u)}{k \cdot (4q)^k}\right) \leq \exp(-k^d). \blacktriangleleft$$

We are ready to prove Theorem 11 (restated below) and thus finish the proof of the reduction.

▶ **Theorem 11.** *Let $k, d, q \in \mathbb{N}$ satisfy $d \geq k \cdot (4q)^{4k \log k}$. Then there exists $\beta \in \mathsf{Schemes}(k, d)$ such that for every q-subset $A \subseteq L(T_{k,d})$ there is a collision with respect to (A, β).*

Proof. We choose the scheme β by picking each bijection uniformly and independently at random. For a fixed q-subset A let C_A denote the event that (A, β) witnesses a collision. In these terms, Lemma 16 says that $\mathbb{P}[\neg C_A] \leq \exp(-k^d)$. Let $\mathcal{A}$ be the family of all q-subsets $A \subseteq L(T_{k,d})$; we have $|\mathcal{A}| \leq 2^{k^d}$. By the union bound, the probability that there exists a q-subset with no collision with β is

$$\mathbb{P}\left[\bigcup_{A \in \mathcal{A}} \neg C_A\right] \leq \sum_{A \in \mathcal{A}} \mathbb{P}[\neg C_A] \leq 2^{k^d} \cdot (1/e)^{k^d} < 1.$$

Consequently, there is a positive probability of choosing a scheme β having a collision with every q-subset. In particular, this means that such a scheme exists. ◀

5 Conclusion

We have shown that no FPT algorithm can achieve an $\mathcal{O}(1)$-approximation for Max Disjoint Paths on acyclic digraphs. However, our reduction blows up the parameter significantly so it does not preserve a running time of the form $f(k)n^{o(k)}$. It is known that such a running time is unlikely for the exact variant of the problem [12]. This leads to a question whether Max Dag Disjoint Paths admits an $\mathcal{O}(1)$-approximation that is faster than $n^{\mathcal{O}(k)}$.

Our proof yields an alternative technique for gap amplification in a parameterized reduction based on the probabilistic method (extending the restricted version appearing in [53]), compared to reductions relying on coding theory [39, 25] or communication complexity [30]. Can this approach come in useful for proving that Parameterized Inapproximability Hypothesis (PIH) follows from FPT≠W[1]?

References

1 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Irrelevant vertices for the planar disjoint paths problem. *J. Comb. Theory, Ser. B*, 122:815–843, 2017. doi:10.1016/j.jctb.2016.10.001.

2 Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with Congestion in Acyclic Digraphs. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2016.7.

3 Matthias Bentert, Fedor V. Fomin, and Petr A. Golovach. Tight approximation and kernelization bounds for vertex-disjoint shortest paths. *arXiv*, abs/2402.15348, 2024. doi:10.48550/arXiv.2402.15348.

4 Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *Journal of the ACM (JACM)*, 68(3):1–40, 2021. doi:10.1145/3444942.

5 Dario Giuliano Cavallaro, Ken-ichi Kawarabayashi, and Stephan Kreutzer. Edge-disjoint paths in eulerian digraphs. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 704–715. ACM, 2024. doi:10.1145/3618260.3649758.

6 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-Exponential Time Hypothesis to fixed parameter tractable inapproximability: Clique, dominating set, and more. *SIAM J. Comput.*, 49(4):772–810, 2020. doi:10.1137/18M1166869.

7 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Pre-reduction graph products: Hardnesses of properly learning DFAs and approximating EDP on dags. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 444–453. IEEE, 2014. doi:10.1109/FOCS.2014.54.

8 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 71–80. IEEE, 2004. doi:10.1109/FOCS.2004.27.

9 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of computing*, 2(1):137–146, 2006. doi:10.4086/TOC.2006.V002A007.

10 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM J. Comput.*, 39(1):281–301, 2009. doi:10.1137/060674442.

11 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. A note on multiflows and treewidth. *Algorithmica*, 54(3):400–412, 2009. doi:10.1007/S00453-007-9129-Z.

12 Rajesh Chitnis. A tight lower bound for edge-disjoint paths on planar dags. *SIAM Journal on Discrete Mathematics*, 37(2):556–572, 2023. doi:10.1137/21M1395089.

13 Kyungjin Cho, Eunjin Oh, and Seunghyeok Oh. Parameterized algorithm for the disjoint path problem on planar graphs: Exponential in k^2 and linear in n. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3734–3758. SIAM, 2023. doi:10.1137/1.9781611977554.CH144.

14 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. *SIAM J. Comput.*, 51(2):17–189, 2022. doi:10.1137/17M1146580.

15 Julia Chuzhoy, David HK Kim, and Shi Li. Improved approximation for node-disjoint paths in planar graphs. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 556–569, 2016. doi:10.1145/2897518.2897538.

16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 197–206, 2013. doi:10.1109/FOCS.2013.29.

18 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

19 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. doi:10.1145/1236457.1236459.

20 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.SWAT.2016.15.

21 Andreas Emil Feldmann, Karthik C S, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/A13060146.

22 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

23 András Frank. Packing paths, cuts, and circuits - a survey. *Paths, Flows and VLSI-Layout*, 49:100, 1990.

24 Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-joung Kwon. Directed tangle tree-decompositions and applications. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 377–405. SIAM, 2022. doi:10.1137/1.9781611977073.19.

25 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Parameterized inapproximability hypothesis under exponential time hypothesis. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 24–35. ACM, 2024. doi:10.1145/3618260.3649771.

26 Venkatesan Guruswami, Xuandi Ren, and Sai Sandeep. Baby PIH: Parameterized Inapproximability of Min CSP. In Rahul Santhanam, editor, *39th Computational Complexity Conference (CCC 2024)*, volume 300 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2024.27.

27 Pinar Heggernes, Pim van't Hof, Erik Jan van Leeuwen, and Reza Saei. Finding disjoint paths in split graphs. *Theory of Computing Systems*, 57:140–159, 2015. doi:10.1007/S00224-014-9580-6.

28 Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, pages 286–295, 1983.

29 Karthik C. S. and Subhash Khot. Almost polynomial factor inapproximability for parameterized k-Clique. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CCC.2022.6.

30 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.

31 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 70–78, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591876.

32 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. doi:10.1016/J.JCTB.2011.07.004.

33 Ken-ichi Kawarabayashi and Stephan Kreutzer. The directed grid theorem. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 655–664, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746586.

34 Jon M. Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *J. Comput. Syst. Sci.*, 57(1):61–73, 1998. doi:10.1006/JCSS.1998.1579.

35 Stavros G Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99(1):63–87, 2004. doi:10.1007/S10107-002-0370-6.

36 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. *arXiv*, abs/2404.03958, 2024 (to appear at FOCS 2024). doi:10.48550/arXiv.2404.03958.

37 Mark R. Kramer and Jan van Leeuwen. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. *Advances in Computing Research*, 2:129–146, 1984.

38 Michael Lampis and Manolis Vasilakis. Parameterized maximum node-disjoint paths. *arXiv*, abs/2404.14849, 2024. doi:10.48550/arXiv.2404.14849.

39 Bingkai Lin. Constant approximating k-Clique is W[1]-hard. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1749–1756. ACM, 2021. doi:10.1145/3406325.3451016.

40 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Constant approximating parameterized k-SetCover is W[2]-hard. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3305–3316. SIAM, 2023. doi:10.1137/1.9781611977554.CH126.

41 Daniel Lokshtanov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1307–1316, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384250.

42 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200, 2020. doi:10.1137/1.9781611975994.134.

43 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-Cut. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 798–809. IEEE, 2020. doi:10.1109/FOCS46700.2020.00079.

44 James F Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3):31–36, 1975.

45 Sridhar Natarajan and Alan P Sprague. Disjoint paths in circular arc graphs. *Nordic Journal of Computing*, 3(3):256–270, 1996.

46 Richard G. Ogier, Vladislav Rutenburg, and Nachum Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE Trans. Inf. Theory*, 39(2):443–455, 1993. doi:10.1109/18.212275.

47 Naoto Ohsaka. On the parameterized intractability of determinant maximization. *Algorithmica*, pages 1–33, 2024. doi:10.1007/S00453-023-01205-0.

48 Bruce Reed. Rooted routing in the plane. *Discrete Applied Mathematics*, 57(2-3):213–227, 1995. doi:10.1016/0166-218X(94)00104-L.

49 Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

50 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

51 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discret. Math.*, 24(1):146–157, 2010. doi:10.1137/070697781.

52 Anand Srinivas and Eytan H. Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wirel. Networks*, 11(4):401–417, 2005. doi:10.1007/S11276-005-1765-0.

53 Michał Włodarczyk. Parameterized inapproximability for steiner orientation by gap amplification. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 104:1–104:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.104.

54 Michał Włodarczyk and Meirav Zehavi. Planar disjoint paths, treewidth, and kernels. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 649–662. IEEE, 2023. doi:10.1109/FOCS57990.2023.00044.

Does Subset Sum Admit Short Proofs?

Michał Włodarczyk
University of Warsaw, Poland

Abstract

We investigate the question whether Subset Sum can be solved by a polynomial-time algorithm with access to a certificate of length poly(k) where k is the maximal number of bits in an input number. In other words, can it be solved using only few nondeterministic bits?

This question has motivated us to initiate a systematic study of certification complexity of parameterized problems. Apart from Subset Sum, we examine problems related to integer linear programming, scheduling, and group theory. We reveal an equivalence class of problems sharing the same hardness with respect to having a polynomial certificate. These include Subset Sum and Boolean Linear Programming parameterized by the number of constraints. Secondly, we present new techniques for establishing lower bounds in this regime. In particular, we show that Subset Sum in permutation groups is at least as hard for nondeterministic computation as 3Coloring in bounded-pathwidth graphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases subset sum, nondeterminism, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.58

Related Version *Full Version*: `https://arxiv.org/abs/2409.03526` [92]

Funding *Michał Włodarczyk*: Supported by Polish National Science Centre SONATA-19 grant number 2023/51/D/ST6/00155.

1 Introduction

Nondeterminism constitutes a powerful lens for studying complexity theory. The most prominent instantiation of this concept is the class NP capturing all problems with solutions checkable in polynomial time. Another well-known example is the class NL of problems that can be solved nondeterministically in logarithmic space [7]. But the usefulness of nondeterminism is not limited to merely filtering candidates for deterministic classes. A question studied in proof complexity theory is how much nondeterminism is needed to solve certain problems or, equivalently, how long proofs have to be to prove certain theorems [27]. Depending on the considered logic, these theorems may correspond to instances of problems complete for NP [65], coNP [28], or W[SAT] [31]. The central goal of proof complexity is to establish lower bounds for increasingly powerful proof systems in the hope of building up techniques to prove, e.g., NP $\neq$ coNP. What is more, there are connections between nondeterministic running time lower bounds and fine-grained complexity [25]. In the context of online algorithms, nondeterminism is used to measure how much knowledge of future requests is needed to achieve a certain performance level [14, 18, 35].

Bounded nondeterminism plays an important role in organizing parameterized complexity theory. The first class studied in this context was W[P] comprising parameterized problems solvable in FPT time, i.e., $f(k) \cdot \text{poly}(n)$, when given access to $f(k) \cdot \log(n)$ nondeterministic bits [29]. In the last decade, classes defined by nondeterministic computation in limited space have attracted significant attention [5, 43, 81] with a recent burst of activity around the class XNLP [15, 16, 17] of problems solvable in nondeterministic time $f(k) \cdot \text{poly}(n)$ and space $f(k) \cdot \log(n)$. While the study of W[P] and XNLP concerns problems considered very hard from the perspective of FPT algorithms, a question that has eluded a systematic examination so far

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 58; pp. 58:1–58:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is **how much nondeterminism is necessary to solve FPT problems in polynomial time.** A related question has been asked about the amount of nondeterminism needed to solve d-CNF-SAT in sub-exponential time [32].

To concretize our question, we say that a parameterized problem P admits a *polynomial certificate* if an instance (I, k) can be solved in polynomial time when given access to poly(k) nondeterministic bits[1]. For example, every problem in NP admits a polynomial certificate under parameterization by the input length. This definition captures, e.g., FPT problems solvable via branching as a certificate can provide a roadmap for the correct branching choices. Furthermore, every parameterized problem that is in NP and admits a polynomial kernelization has a polynomial certificate given by the NP certificate for the compressed instance. The containment in NP plays a subtle role here: Wahlström [90] noted that a polynomial compression for the K-Cycle problem is likely to require a target language from outside NP exactly because K-Cycle does not seem to admit a polynomial certificate.

In this article we aim to organize the folklore knowledge about polynomial certification into a systematic study, provide new connections, techniques, and motivations, and lay the foundations for a hardness framework.

When is certification easy or hard? The existence of a certificate of size $p(k) = \text{poly}(k)$ entails an FPT algorithm with running time $2^{p(k)}\text{poly}(n)$, by enumerating all possible certificates. We should thus restrict ourselves only to problems solvable within such running time. On the other hand, when such an algorithm is available then one can solve the problem in polynomial time whenever $p(k) \le \log n$. Therefore, it suffices to handle the instances with $\log n < p(k)$. Consequently, for such problems it is equivalent to ask for a certificate of size poly$(k + \log n)$ as this can be bounded polynomially in k via the mentioned trade-off (see Lemma 11). This observation yields polynomial certificates for problems parameterized by the solution size, such as Multicut [74] or Planarization [56], which do not fall into the previously discussed categories.

What are the problems solvable in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$ yet unlikely to admit a polynomial certificate? The Bandwidth problem has been conjectured not be belong to W[P] because one can merge multiple instances into one, without increasing the parameter, in such a way that the large instance is solvable if and only if all the smaller ones are. It is conceivable that a certificate for the large instance should require at least one bit for each of the smaller instances, hence it cannot be short [45]. The same argument applies to every parameterized problem that admits an AND-composition, a construction employed to rule out polynomial kernelization [33, 47], which effectively encodes a conjunction of multiple 3SAT instances as a single instance of the problem. Such problems include those parameterized by graph width measures like treewidth or pathwidth, and it is hard to imagine polynomial certificates for them. However, kernelization hardness can be also established using an OR-composition, which does not stand at odds with polynomial certification.

The close connection between AND-composition and polynomial certificates has been observed by Drucker, Nederlof, and Santhanam [39] who focused on parameterized search problems solvable by *One-sided Probabilistic Polynomial* (OPP) algorithms (cf. [80]). They asked which problems admit an OPP algorithm that finds a solution with probability $2^{-\text{poly}(k)}$.

[1] It is more accurate to say that a "certificate" refers to a particular instance while a problem can admit a "certification". We have decided however to choose a shorter and more established term. We also speak of "certificates" instead of "witnesses" because "witness" sometimes refers to a concrete representation of a solution for problems in NP, see e.g., [39].

This may seem much more powerful than using poly(k) nondeterministic bits but the success probability can be replaced by $\Omega(1)$ when given a single access to an oracle solving k-variable Circuit-SAT [39, Lemma 3.5]. A former result of Drucker [37] implies that an OPP algorithm with success probability $2^{-\mathrm{poly}(k)}$ (also called a *polynomial Levin witness compression*) for a search problem admitting a so-called constructive AND-composition would imply NP $\subseteq$ coNP/poly [39, Theorem 3.2].

Constructive vs. non-constructive proofs. The restriction to search problems is crucial in the work [39] because the aforementioned hardness result does not apply to algorithms that may recognize yes-instances without constructing a solution explicitly but by proving its existence in a non-constructive fashion. As noted by Drucker [38, §1.3], his negative results do not allow to rule out this kind of algorithms.

In general, search problems may be significantly harder than their decision counterparts. For example, there are classes of search problems for which the solution is always guaranteed to exist (e.g., by the pigeonhole principle, in the case of class PPP [1]) but the existence of a polynomial algorithm computing some solution is considered unlikely. For a less obvious example, consider finding a non-trivial divisor of a given integer n. A polynomial (in $\log n$) algorithm finding a solution could be used to construct the factorization of n, resolving a major open problem. But the existence of a solution is equivalent to n being composite and this can be verified in polynomial time by the AKS primality test [4].

As yet another example, consider the problem of finding a knotless embedding of a graph, i.e., an embedding in $\mathbb{R}^3$ in which every cycle forms a trivial knot in a topological sense. The class $\mathcal{G}$ of graphs admitting such an embedding is closed under taking minors so Robertson and Seymour's Theorem ensures that $\mathcal{G}$ is characterized by a finite set of forbidden minors [85], leading to a polynomial algorithm for recognizing graphs from $\mathcal{G}$. Observe that excluding all the forbidden minors yields a non-constructive proof that a knotless embedding exists. On the other hand, the existence of a polynomial algorithm constructing such an embedding remains open [71].

To address this discrepancy, we propose the following conjecture which asserts that not only *finding* assignments to many instances of 3SAT requires many bits of advice but even *certifying* that such assignments exist should require many bits of advice. We define the parameterized problem AND-3SAT[k] where an instance consists of a sequence of n many 3SAT formulas on k variables each, and an instance belongs to the language if all these formulas are satisfiable. We treat k as a parameter.

▶ **Conjecture 1.** AND-3SAT[k] *does not admit a polynomial certificate unless NP* $\subseteq$ *coNP/poly.*

Observe that AND-3SAT[k] is solvable in time $2^k \cdot \mathrm{poly}(k) \cdot n$, so the questions whether it admits a certificate of size poly(k), poly(k) $\cdot \log n$, or poly($k + \log n$) are equivalent. We formulate Conjecture 1 as a conditional statement because we believe that its proof in the current form is within the reach of the existing techniques employed in communication complexity and kernelization lower bounds [33, 80]. Then the known examples of AND-composition could be interpreted as reductions from AND-3SAT[k] that justify non-existence of polynomial certificates.

1.1 The problems under consideration

Our focus: Subset Sum. In Subset Sum we are given a sequence of n integers (also called items), a target integer t (all numbers encoded in binary), and we ask whether there is a subsequence summing up to t. This is a fundamental NP-hard problem that can be solved in pseudo-polynomial time $\mathcal{O}(tn)$ by the classic algorithm by Bellman from the 50s [11]. In 2017 the running time has been improved to $\tilde{\mathcal{O}}(t+n)$ by Bringmann [19]. Subset Sum reveals miscellaneous facets in complexity theory: it has been studied from the perspective of exponential algorithms [77, 78], logarithmic space [59, 61], approximation [21, 26, 63, 76], kernelization [36, 52, 57], fine-grained complexity [3, 22, 83, 82], cryptographic systems [55], and average-case analysis [75]. Our motivating goal is the following question.

▶ **Question 2.** *Does* Subset Sum *admit a polynomial certificate for parameter* $k = \log t$*?*

From this point of view, the pseudo-polynomial time $\mathcal{O}(tn)$ can be interpreted as FPT running time $\mathcal{O}(2^k n)$. It is also known that a kernelization of size poly(k) is unlikely [36]. Observe that we cannot hope for a certificate of size $o(\log t)$ because the algorithm enumerating all possible certificates would solve Subset Sum in time $2^{o(\log t)} n^{\mathcal{O}(1)} = t^{o(1)} n^{\mathcal{O}(1)}$ contradicting the known lower bound based on the Exponential Time Hypothesis (ETH) [3].

The parameterization by the number of relevant bits exhibits a behavior different from those mentioned so far, that is, width parameters and solution size, making it an uncharted territory for nondeterministic algorithms. The study of Subset Sum$[\log t]$ was suggested by Drucker et al. [39] in the context of polynomial witness compression. These two directions are closely related yet ultimately incomparable: the requirement to return a solution makes the task more challenging but the probabilistic guarantee is less restrictive than constructing a certificate. However, establishing hardness in both paradigms boils down to finding a reduction of a certain kind from AND-3SAT$[k]$.

The *density* of a Subset Sum instance is defined as $n \,/\, \log(t)$ in the cryptographic context [8, 54, 69]. As it is straightforward to construct a certificate of size n, we are mostly interested in instances of high density, which also appear hard for exponential algorithms [8]. On the other hand, instances that are very dense enjoy a special structure that can be leveraged algorithmically [22, 49]. The instances that seem the hardest in our regime are those in which n is slightly superpolynomial in $\log t$. Apart from the obvious motivation to better understand the structure of Subset Sum, we believe that the existence of short certificates for dense instances could be valuable for cryptography.

There are several other studied variants of the problem. In Unbounded Subset Sum the input is specified in the same way but one is allowed to use each number repeatedly. Interestingly, this modification enables us to certify a solution with $\mathcal{O}(\log^2 t)$ bits (see the full version). Another variant is to replace the addition with some group operation. In Group-G Subset Sum we are given a sequence of n elements from G and we ask whether one can pick a subsequence whose group product equals the target element $t \in G$. Note that we do not allow to change the order of elements when computing the product what makes a difference for non-commutative groups. This setting has been mostly studied for G being the cyclic group $\mathbb{Z}_q$ [9, 10, 24, 66, 84]. To capture the hardness of an instance, we choose the parameter to be $\log |G|$ (or equivalent). In particular, we will see that Group-$\mathbb{Z}_q$ Subset Sum$[\log q]$ is equivalent in our regime to Subset Sum$[\log t]$. We will also provide examples of groups for which certification is either easy or conditionally hard.

Integer Linear Programming. We shall consider systems of equations in the form $\{Ax = b \mid x \in \{0,1\}^n\}$ where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, with the parameter being the number of constraints m. This is a special case of Integer Linear Programming (ILP) over boolean domain, known in the literature as *pseudo-boolean optimization.* This case has been recognized as particularly interesting by the community working on practical ILP solvers because pseudo-boolean optimization can be treated with SAT solvers [6, 23, 40, 60, 89]. It is also applicable in the fields of approximation algorithms [91] and election systems [72]. Eisenbrand and Weismantel [42] found an elegant application of Steinitz Lemma to this problem and gave an FPT algorithm with running time $(||A||_\infty + m)^{\mathcal{O}(m^2)} \cdot n$ (cf. [58]).

For simplicity we will consider variants with bounded $||A||_\infty$. In the 0-1 ILP problem we restrict ourselves to matrices $A \in \{-1, 0, 1\}^{m \times n}$ and in Monotone 0-1 ILP we consider $A \in \{0, 1\}^{m \times n}$. A potential way to construct a short certificate would be to tighten the *proximity bounds* from [42] for such matrices: find an extremal solution x^* to the linear relaxation $\{Ax = b \mid x \in [0, 1]^n\}$ and hope that some integral solution z lies nearby, i.e., $||z - x^*||_1 \le \text{poly}(m)$. This however would require tightening the bounds on the vector norms in the *Graver basis* of the matrix A. Unfortunately, there are known lower bounds making this approach hopeless [12, 13, 67].

It may be tempting to seek the source of hardness in the large values in the target vector b. We will see however that the problem is no easier when we assume $b = 0$ and look for any non-zero solution. We refer to such problem as 0-Sum 0-1 ILP.

A special case of Monotone 0-1 ILP is given by a matrix $A \in \{0, 1\}^{m \times n}$ with $n = \binom{m}{d}$ columns corresponding to all size-d subsets of $\{1, \dots, m\}$. Then $Ax = b$ has a boolean solution if and only if b forms a degree sequence of some d-hypergraph, i.e., it is d-hypergraphic. There is a classic criterion by Erdős for a sequence to be graphic [44] (i.e., 2-hypergraphic) but already for $d = 3$ deciding if b is d-hypergraphic becomes NP-hard [34]. It is straightforward to certify a solution with m^d bits, what places the problem in NP for each fixed d, but it is open whether it is in NP when d is a part of the input (note that the matrix A is implicit so the input size is $\mathcal{O}(md \log m)$). This basically boils down to the same dilemma: **can we certify the existence of a boolean ILP solution x without listing x in its entirety?**

There is yet another motivation to study the certification complexity of 0-1 ILP. If this problem admits a certificate of size poly(m) then any other problem that can be modeled by 0-1 ILP with few constraints must admit a short certificate as well. This may help classifying problems into these that can be solved efficiently with ILP solvers and those that cannot.

Other problems parameterized by the number of relevant bits. A classic generalization of Subset Sum is the Knapsack problem where each item is described by a size p_i and a weight w_i; here we ask for a subset of items of total weight at least w but total size not exceeding t. Following Drucker et al. [39] we parameterize it by the number of bits necessary to store items' sizes and weights, i.e., $\log(t+w)$. The need to process weights makes Knapsack harder than Subset Sum from the perspective of fine-grained complexity [30] but they are essentially equivalent on the ground of exponential algorithms [77]. We will see that they are equivalent in our regime as well.

We will also consider the following scheduling problem which is in turn a generalization of Knapsack. In Scheduling Weighted Tardy Jobs we are given a set of n jobs, where each job $j \in [n]$ has a processing time $p_j \in \mathbb{N}$, a weight $w_j \in \mathbb{N}$, and a due date $d_j \in \mathbb{N}$. We schedule the jobs on a single machine and we want to minimize the total weight of jobs completed after their due dates (those jobs are called *tardy*). Equivalently, we try to maximize the total weight of jobs completed in time.

In the scheduling literature, Scheduling Weighted Tardy Jobs is referred to as $1||\sum w_j U_j$ using Graham's notation. The problem is solvable in pseudo-polynomial time $\mathcal{O}(n \cdot d_{max})$ by the classic Lawler and Moore's algorithm [70]. The interest in $1||\sum w_j U_j$ has been revived due to the recent advances in fine-grained complexity [2, 20, 46, 64]. Here, the parameter that captures the number of relevant bits is $\log(d_{max} + w_{max})$.

1.2 Our contribution

The standard *polynomial parameter transformation* (PPT) is a polynomial-time reduction between parameterized problems that maps an instance with parameter k to one with parameter $k' = \text{poly}(k)$. We introduce the notion of a *nondeterministic polynomial parameter transformation* (NPPT) which extends PPT by allowing the reduction to guess poly(k) nondeterministic bits. Such reductions preserve the existence of a polynomial certificate. We write $P \leq_{\text{NPPT}} Q$ (resp. $P \leq_{\text{PPT}} Q$) to indicate that P admits a NPPT (resp. PPT) into Q.

We demonstrate how NPPT help us organize the theory of polynomial certification, similarly as PPT come in useful for organizing the theory of Turing kernelization [53]. As our first result, we present an equivalence class of problems that share the same certification-hardness status as Subset Sum[$\log t$]. In other words, either all of them admit a polynomial certificate or none of them. Despite apparent similarities between these problems, some of the reductions require a nontrivial use of nondeterminism.

▶ **Theorem 3.** *The following parameterized problems are equivalent with respect to NPPT:*

1. Subset Sum[$\log t$]
2. Knapsack[$\log(t + w)$], Knapsack[$\log(p_{max} + w_{max})$]
3. 0-1 ILP[m], Monotone 0-1 ILP[m], 0-Sum 0-1 ILP[m]
4. Group-$\mathbb{Z}_q$ Subset Sum[$\log q$]

Even though we are unable to resolve Question 2, we believe that revealing such an equivalence class supports the claim that a polynomial certificate for Subset Sum[$\log t$] is unlikely. Otherwise, there must be some intriguing common property of all problems listed in Theorem 3 that has eluded researchers so far despite extensive studies in various regimes.

Next, we present two negative results. They constitute a proof of concept that AND-3SAT[k] can be used as a non-trivial source of hardness. First, we adapt a reduction from [2] to show that scheduling with weights and due dates is hard assuming Conjecture 1.

▶ **Theorem 4 (★).** AND-3SAT[k] $\leq_{PPT}$ Scheduling Weighted Tardy Jobs[$\log(d_{max} + w_{max})$].

It is possible to formulate this result in terms of AND-composition but we chose not to work with this framework since it is tailored for refuting kernelization and relies on concepts that do not fit into our regime (e.g., polynomial relation [47]).

Our second hardness result involves Group-S_k Subset Sum[k]: a variant of Subset Sum on permutation groups. Such groups contain exponentially-large cyclic subgroups (see Lemma 13) so this problem is at least as hard as Group-$\mathbb{Z}_q$ Subset Sum[$\log q$] (which is equivalent to Subset Sum[$\log t$]). We reduce from 3Coloring parameterized by pathwidth which is at least as hard as AND-3SAT[k] with respect to PPT. Indeed, we can transform each 3SAT formula in the input (each of size $\mathcal{O}(k^3)$) into an instance of 3Coloring via the standard NP-hardness proof, and take the disjoint union of such instances, which implies AND-3SAT[k] $\leq_{\text{PPT}}$ 3Coloring[pw]. Notably, the reduction in the other direction is unlikely (see the full version) so 3Coloring[pw] is probably harder than AND-3SAT[k].

▶ **Theorem 5.** 3Coloring[pw] $\leq_{NPPT}$ Group-S_k Subset Sum[k].

Consequently, Group-S_k Subset Sum[k] does not admit a polynomial certificate assuming Conjecture 1 and NP $\not\subseteq$ coNP/poly. Unlike Theorem 4, this time establishing hardness requires a nondeterministic reduction. An interesting feature of 3Coloring[pw] is that it is NL-complete under logspace reductions when the pathwidth pw is restricted to $\mathcal{O}(\log n)$ [5, 81]. On the other hand, Subset Sum can be solved in time $\tilde{\mathcal{O}}(tn^2)$ and space polylog(tn) using algebraic techniques [59]. Therefore, obtaining a logspace PPT from 3Coloring[pw] to Subset Sum[$\log t$] (where $\textsc{pw} = \log n$ implies $t = 2^{\text{polylog}(n)}$) would lead to a surprising consequence: a proof that NL $\subseteq$ DSPACE(polylog(n)) that is significantly different from Savitch's Theorem (see also discussion in [81, §1] on low-space determinization). This suggests that a hypothetical reduction to Subset Sum[$\log t$] should either exploit the "full power" of NPPT (so it cannot be improved to a logspace PPT) or start directly from AND-3SAT[k].

Finally, we examine the case of the group family $\mathbb{Z}_k^k$ on which Subset Sum is still NP-hard (as this generalizes Subset Sum on cyclic groups) but enjoys a polynomial certificate. Specifically, we exploit the bound on the maximal order of an element in $\mathbb{Z}_k^k$ to prove that there always exists a solution of bounded size.

▶ **Lemma 6 (★).** Group-$\mathbb{Z}_k^k$ Subset Sum[k] *admits a polynomial certificate.*

In summary, Group-G Subset Sum appears easy for $G = \mathbb{Z}_k^k$ (due to bounded maximal order), hard for $G = S_k$ (due to non-commutativity), and the case $G = \mathbb{Z}_{2^k}$ lies somewhere in between. In the light of Theorem 3, tightening this gap seems a promising avenue to settle Question 2.

Organization of the paper. We begin with the preliminaries where we formally introduce the novel concepts, such as NPPT. We prove Theorems 3 and 5 in Sections 3 and 4, respectively. The proofs marked with (★) can be found in the full version of the article [92].

2 Preliminaries

We denote the set $\{1, \dots, n\}$ by $[n]$. For a sequence $x_1, x_2, \dots, x_n$, its subsequence is any sequence of the form $x_{i_1}, \dots, x_{i_m}$ for some choice of increasing indices $1 \le i_1 < \dots < i_m \le n$. All considered logarithms are binary.

A parameterized problem P is formally defined as a subset of $\Sigma^* \times \mathbb{N}$. For the sake of disambiguation, whenever we refer to a parameterized problem, we denote the choice of the parameter in the $[\cdot]$ bracket, e.g., 3Coloring[pw]. We call P *fixed-parameter tractable* (FPT) is the containment $(I, k) \in P$ can be decided in time $f(k) \cdot \text{poly}(|I|)$ for some computable function f. We say that P admits a *polynomial compression* into a problem Q if there is a polynomial-time algorithm that transform (I, k) into an equivalent instance of Q of size poly(k). If Q coincides with the non-parameterized version of P then such an algorithm is called a *polynomial kernelization.* A *polynomial Turing kernelization* for P is a polynomial-time algorithm that determines if $(I, k) \in P$ using an oracle that can answer if $(I', k') \in P$ whenever $|I'| + k' \le \text{poly}(k)$.

▶ **Definition 7.** *Let $P \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that P has a polynomial certificate if there is an algorithm $\mathcal{A}$ that, given an instance (I, k) of P and a string y of poly(k) bits, runs in polynomial time and accepts or rejects (I, k) with the following guarantees.*

1. *If $(I, k) \in P$, then there exists y for which $\mathcal{A}$ accepts.*
2. *If $(I, k) \notin P$, then $\mathcal{A}$ rejects (I, k) for every y.*

▶ **Lemma 8.** *Let $P \subseteq \Sigma^* \times \mathbb{N}$ and $Q \subseteq \Sigma^*$. Suppose that $Q \in NP$ and P admits a polynomial compression into Q. Then P admits a polynomial certificate.*

Proof. For a given instance (I, k) of P we execute the compression algorithm to obtain an equivalent instance I' of Q of size poly(k). Since $Q \in$ NP the instance I' can be solved in polynomial-time with an access to a string y of poly$(|I'|) =$ poly(k) nondeterministic bits. Then y forms a certificate for (I, k). ◀

▶ **Definition 9.** *Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems. An algorithm $\mathcal{A}$ is called a polynomial parameter transformation (PPT) from P to Q if, given an instance (I, k) of P, runs in polynomial time, and outputs an equivalent instance (I', k') of Q with $k' \le$ poly(k).*

An algorithm $\mathcal{B}$ is called a nondeterministic polynomial parameter transformation (NPPT) from P to Q if, given an instance (I, k) of P and a string y of poly(k) bits, runs in polynomial time, and outputs an instance (I', k') of Q with the following guarantees.

1. *$k' \le$ poly(k)*
2. *If $(I, k) \in P$, then there exists y for which $\mathcal{B}$ outputs $(I', k') \in Q$.*
3. *If $(I, k) \notin P$, then $\mathcal{B}$ outputs $(I', k') \notin Q$ for every y.*

Clearly, PPT is a special case of NPPT. We write $P \le_{\text{PPT}} Q$ ($P \le_{\text{NPPT}} Q$) if there is a (nondeterministic) PPT from P to Q. We write $P \equiv_{\text{PPT}} Q$ ($P \equiv_{\text{NPPT}} Q$) when we have reductions in both directions. It is easy to see that the relation $\le_{\text{NPPT}}$ is transitive. Similarly as the relation $\le_{\text{PPT}}$ is monotone with respect to having a polynomial kernelization, the relation $\le_{\text{NPPT}}$ is monotone with respect to having a polynomial certificate.

▶ **Lemma 10.** *Let $P, Q \in \Sigma^* \times \mathbb{N}$ be parameterized problems. If $P \le_{\text{NPPT}} Q$ and Q admits a polynomial certificate then P does as well.*

Proof. Given an instance (I, k) of P the algorithm guesses a string y_1 of length poly(k) guiding the reduction to Q and constructs an instance (I', k') with $k' =$ poly(k). Then it tries to prove that $(I', k') \in Q$ by guessing a certificate y_2 of length poly$(k') =$ poly(k). ◀

A different property transferred by PPT is polynomial Turing kernelization. Hermelin et al. [53] proposed a hardness framework for this property by considering complexity classes closed under PPT (the WK-hierarchy).

Next, we prove the equivalence mentioned in the Introduction.

▶ **Lemma 11.** *Suppose $P \subseteq \Sigma^* \times \mathbb{N}$ admits an algorithm $\mathcal{A}$ deciding if $(I, k) \in P$ in time $2^{p(k)}$poly$(|I|)$ where $p(\cdot)$ is a polynomial function. Then $P[k] \equiv_{\text{PPT}} P[k + \log |I|]$.*

Proof. The direction $P[k + \log |I|] \le_{\text{PPT}} P[k]$ is trivial. To give a reduction in the second direction, we first check if $p(k) \le \log |I|$. If yes, we execute $\mathcal{A}$ in time poly$(|I|)$ and according to the outcome we return a trivial yes/no-instance. Otherwise we have $\log |I| < p(k)$ so we can output (I, k') for the new parameter $k' = k + \log |I|$ being polynomial in k. ◀

Pathwidth. A path decomposition of a graph G is a sequence $\mathcal{P} = (X_1, X_2, \dots, X_r)$ of *bags*, where $X_i \subseteq V(G)$, and:

1. For each $v \in V(G)$ the set $\{i \mid v \in X_i\}$ forms a non-empty subinterval of $[r]$.
2. For each edge $uv \in E(G)$ there is $i \in [r]$ with $\{u, v\} \subseteq X_i$.

The *width* of a path decomposition is defined as $\max_{i=1}^{r} |X_i| - 1$. The *pathwidth* of a graph G is the minimum width of a path decomposition of G.

▶ **Lemma 12** ([29, Lemma 7.2]). *If a graph G has pathwidth at most p, then it admits a* nice *path decomposition $\mathcal{P} = (X_1, X_2, \dots, X_r)$ of width at most p, for which:*

- *$X_1 = X_r = \emptyset$.*

- *For each $i \in [r-1]$ there is either a vertex $v \notin X_i$ for which $X_{i+1} = X_i \cup \{v\}$ or a vertex $v \in X_i$ for which $X_{i+1} = X_i \setminus \{v\}$.*

Furthermore, given any path decomposition of G, we can turn it into a nice path decomposition of no greater width, in polynomial time.

The bags of the form $X_{i+1} = X_i \cup \{v\}$ are called *introduce bags* while the ones of the form $X_{i+1} = X_i \setminus \{v\}$ are called *forget bags*.

Similarly as in the previous works [5, 16, 81] we assume that a path decomposition of certain width is provided with the input. This is not a restrictive assumption for our model since pathwidth can be approximated within a polynomial factor in polynomial time [50].

Group theory. The basic definitions about groups can be found in the book [86]. A homomorphism between groups G, H is a mapping $\phi\colon G \to H$ that preserves the group operation, i.e., $\phi(x) \circ_H \phi(y) = \phi(x \circ_G y)$ for all $x, y \in G$. An isomorphism is a bijective homomorphism and an automorphism of G is an isomorphism from G to G. We denote by $Aut(G)$ the automorphism group of G with the group operation given as functional composition. A subgroup N of G is *normal* if for every $g \in G, n \in N$ we have $g \circ_G n \circ_G g^{-1} \in N$.

The symmetric group S_k comprises permutations over the set $[k]$ with the group operation given by composition. For a permutation $\pi \in S_k$ we consider a directed graph over the vertex set $[k]$ and arcs given as $\{(v, \pi(v)) \mid v \in [k]\}$. The cycles of this graph are called the cycles of π.

We denote by $\mathbb{Z}_k$ the cyclic group with addition modulo k. We write the corresponding group operation as $\oplus_k$. An order of an element $x \in G$ is the size of the cyclic subgroup of G generated by x. The Landau's function $g(k)$ is defined as the maximum order of an element x in S_k. It is known that $g(k)$ equals $\max \mathsf{lcm}(k_1, \dots, k_\ell)$ over all partitions $k = k_1 + \dots + k_\ell$ (these numbers correspond to the lengths of cycles in x) and that $g(k) = 2^{\Theta(\sqrt{k \log k})}$ [79]. An element of large order can be found easily if we settle for a slightly weaker bound.

▶ **Lemma 13.** *For each k there exists $\pi \in S_k$ of order $2^{\Omega(\sqrt{k}/\log k)}$ and it can be found in time $poly(k)$.*

Proof. Consider all the primes $p_1, \dots, p_\ell$ that are smaller than $\sqrt{k}$. By the prime number theorem there are $\ell = \Theta(\sqrt{k}/\log k)$ such primes [51]. We have $p_1 + \dots + p_\ell \le \sqrt{k} \cdot \sqrt{k} = k$ so we can find a permutation in S_k with cycles of lengths $p_1, \dots, p_\ell$ (and possibly trivial cycles of length 1). We have $\mathsf{lcm}(p_1, \dots, p_\ell) = \prod_{i=1}^{\ell} p_i \ge 2^\ell = 2^{\Omega(\sqrt{k}/\log k)}$. ◀

For two groups N, H and a homomorphism $\phi\colon H \to Aut(N)$ we define the *outer semidirect product* [86] $N \rtimes_\phi H$ as follows. The elements of $N \rtimes_\phi H$ are $\{(n, h) \mid n \in N, h \in H\}$ and the group operation $\circ$ is given as $(n_1, h_1) \circ (n_2, h_2) = (n_1 \circ \phi_{h_1}(n_2), h_1 \circ h_2)$. A special case of the semidirect product occurs when we combine subgroups of a common group.

▶ **Lemma 14** ([86, §4.3])**.** *Let G be a group with a normal subgroup N and a subgroup H, such that every element $g \in G$ can be written uniquely as $g = n \circ h$ for $n \in N, h \in H$. Let $\phi\colon H \to Aut(N)$ be given as $\phi_h(n) = h \circ n \circ h^{-1}$ (this is well-defined because N is normal in G). Then G is isomorphic to the semidirect product $N \rtimes_\phi H$.*

Group-G Subset Sum **Parameter:** $\log |G|$
Input: A sequence of elements $g_1, g_2, \dots, g_n \in G$, an element $g \in G$
Question: Is there a subsequence $(i_1 < i_2 < \dots < i_r)$ of $[n]$ such that $g_{i_1} \circ g_{i_2} \circ \dots \circ g_{i_r} = g$?

We assume that the encoding of the group elements as well as the group operation $\circ$ are implicit for a specific choice of a group family. For a group family parameterized by k, like $(S_k)_{k=1}^{\infty}$, we treat k as the parameter. In all considered cases it holds that $k \leq \log|G| \leq \text{poly}(k)$ so these two parameterizations are equivalent under PPT.

3 Equivalences

We formally introduce the variants of ILP that will be studied in this section.

0-1 ILP **Parameter:** m
Input: A matrix $A \in \{-1,0,1\}^{m\times n}$, a vector $b \in \mathbb{Z}^m$
Question: Is there a vector $x \in \{0,1\}^n$ for which $Ax = b$?

In Monotone 0-1 ILP we restrict ourselves to matrices $A \in \{0,1\}^{m\times n}$. In 0-Sum 0-1 ILP we have $A \in \{-1,0,1\}^{m\times n}$ and we seek a binary vector $x \neq 0$ for which $Ax = 0$.

We first check that all the parameterized problems considered in this section are solvable in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$**.** For Subset Sum$[\log t]$ we can use the classic $\mathcal{O}(tn)$-time algorithm [11] which can be easily modified to solve Group-$\mathbb{Z}_q$ Subset Sum$[\log q]$ in time $\mathcal{O}(qn)$. For Knapsack$[\log(p_{max} + w_{max})]$ there is an $\mathcal{O}(p_{max} \cdot w_{max} \cdot n)$-time algorithm [82] which also works for the larger parameterization by $\log(t + w)$. Next, 0-1 ILP$[m]$ can be solved in time $2^{\mathcal{O}(m^2 \log m)} \cdot n$ using the algorithm for general matrix A [42]. This algorithm can be used to solve 0-sum 0-1 ILP$[m]$ due to Observation 21. Hence by Lemma 11 we can assume in our reductions that $(\log n)$ is bounded by a polynomial function of the parameter.

▶ **Lemma 15.** Knapsack$[\log(t + w)] \equiv_{PPT}$ Knapsack$[\log(p_{max} + w_{max})]$.

Proof. We only need to show the reduction from Knapsack$[\log(p_{max} + w_{max})]$. When $p_{max} \cdot n < t$ we can afford taking all the items. On the other hand, if $w_{max} \cdot n < w$ then no solution can exist. Therefore, we can assume that $\log t \leq \log p_{max} + \log n$ and $\log w \leq \log w_{max} + \log n$. By Lemma 11 we can assume $\log n$ to be polynomial in $\log(p_{max} + w_{max})$ so the new parameter $\log(t + w)$ is polynomial in the original one. ◀

▶ **Lemma 16.** Subset Sum$[\log t] \equiv_{NPPT}$ Knapsack$[\log(t + w)]$.

Proof. The $(\leq)$ reduction is standard: we translate each input integer p_i into an item (p_i, p_i) and set $w = t$. Then we can pack items of total weight t into a knapsack of capacity t if and only if the Subset Sum instance is solvable.

Now consider the $(\geq)$ reduction. Let $k = \log(t+w)$. By the discussion at the beginning of this section we can assume that $\log n \leq \log(t \cdot w) \leq 2k$. We can also assume that $w_{max} < w$ as any item with weight exceeding w and size fitting into the knapsack would form a trivial solution. Let $W = w \cdot n + 1$.

Suppose there is a set of items with total size equal $t' \leq t$ and total weight equal $w' \geq w$. Note that w' must be less than W. We nondeterministically choose t' and w': this requires guessing $\log t + \log W \leq 4k$ bits. Now we create an instance of Subset Sum by mapping each item (p_i, w_i) into integer $p_i \cdot W + w_i$ and setting the target integer to $t'' = t' \cdot W + w'$. If we guessed (t', w') correctly then such an instance clearly has a solution. On the other hand, if this instance of Subset Sum admits a solution then we have $\sum_{i\in I}(p_i \cdot W + w_i) = t' \cdot W + w'$ for some $I \subseteq [n]$. Since both w' and $\sum_{i\in I'} w_i$ belong to $[1, W)$ we must have $\sum_{i\in I} w_i = w'$ and $\sum_{i\in I} p_i = t'$ so the original instance of Knapsack has a solution as well. Finally, it holds that $\log t'' \leq 5k$ so the parameter is being transformed linearly. ◀

We will need the following extension of the last argument.

▶ **Lemma 17.** *Let $W \in \mathbb{N}$ and $a_1, \dots, a_n$, $b_1, \dots, b_n$ be sequences satisfying $a_i, b_i \in [0, W)$ for each $i \in [n]$. Suppose that $S := \sum_{i=1}^n a_i W^{i-1} = \sum_{i=1}^n b_i W^{i-1}$. Then $a_i = b_i$ for each $i \in [n]$.*

Proof. Consider the remainder of S when divided by W. Since W divides all the terms in S for $i \in [2, n]$ and $a_1, b_1 \in [0, W)$ we must have $a_1 = (S \mod W) = b_1$. Next, consider $S' = (S - a_1)/W = \sum_{i=2}^n a_i W^{i-2} = \sum_{i=2}^n b_i W^{i-2}$. Then $a_2 = (S' \mod W) = b_2$. This argument generalizes readily to every $i \in [n]$. ◀

▶ **Lemma 18.** Subset Sum$[\log t] \equiv_{NPPT}$ Monotone 0-1 ILP$[m]$.

Proof. ($\le$): Consider an instance $(\{p_1, \dots, p_n\}, t)$ of Subset Sum. Let $k = \lceil \log t \rceil$. We can assume that all numbers p_i belong to the interval $[t]$. For an integer $x \in [t]$ let $\mathsf{bin}(x) \in \{0,1\}^k$ denote the binary encoding of x so that $x = \sum_{j=1}^k \mathsf{bin}(x)_j \cdot 2^{j-1}$. Observe that the condition $x_1 + \cdots + x_m = t$ can be expressed as $\sum_{j=1}^k \left(\sum_{i=1}^m \mathsf{bin}(x_i)_j\right) \cdot 2^{j-1} = t$.

We nondeterministically guess a sequence $b = (b_1, \dots, b_k)$ so that b_j equals $\sum_{i \in I} \mathsf{bin}(p_i)_j$ where $I \subseteq [n]$ is a solution. This sequence must satisfy $\max_{j=1}^k b_j \le t$, and so we need k^2 nondeterministic bits to guess b. We check if the sequence b satisfies $\sum_{j=1}^k b_j \cdot 2^{j-1} = t$; if no then the guess was incorrect and we return a trivial no-instance. Otherwise we construct an instance of Monotone 0-1 ILP$[k]$ with a system $Ax = b$. The vector b is given as above and its length is k. The matrix A comprises n columns where the i-th column is $\mathsf{bin}(p_i)$. This system has a solution $x \in \{0,1\}^n$ if and only if there exists $I \subseteq [n]$ so that $\sum_{i \in I} \mathsf{bin}(p_i)_j = b_j$ for all $j \in [k]$. This implies that $\sum_{i \in I} p_i = t$. Conversely, if such a set $I \subseteq [n]$ exists, then there is $b \in [t]^k$ for which $Ax = b$ admits a boolean solution.

($\ge$): Consider an instance $Ax = b$ of Monotone 0-1 ILP$[m]$. As usual, we assume $\log n \le \mathrm{poly}(m)$. We can also assume that $||b||_\infty \le n$ as otherwise $Ax = b$ is clearly infeasible. We construct an instance of Subset Sum with n items and target integer $t = \sum_{j=1}^m b_j \cdot (n+1)^{j-1}$. Note that $t \le m \cdot ||b||_\infty \cdot (n+1)^m$ so $\log t \le \mathrm{poly}(m)$. For $i \in [n]$ let $a^i \in \{0,1\}^m$ denote the i-th column of the matrix A. We define $p_i = \sum_{j=1}^m a^i_j \cdot (n+1)^{j-1}$ and we claim that that instance $J = (\{p_1, \dots, p_n\}, t)$ of Subset Sum is solavble exactly when the system $Ax = b$ has a boolean solution.

First, if $x \in \{0,1\}^m$ forms a solution to $Ax = b$ then for each $j \in [m]$ we have $\sum_{i=1}^n x_i a^i_j (n+1)^{j-1} = b_j (n+1)^{j-1}$ and so $\sum_{i=1}^n x_i p_i = t$. Hence the set $I = \{i \in [n] \mid x_i = 1\}$ encodes a solution to J. In the other direction, suppose that there is $I \subseteq [n]$ for which $\sum_{i \in I} p_i = t$. Then $t = \sum_{j=1}^m (\sum_{i \in I} a^i_j) \cdot (n+1)^{j-1}$. Due to Lemma 17 we must have $b_i = \sum_{i \in I} a^i_j$ for each $i \in [m]$ and there is subset of columns of A that sums up to the vector b. This concludes the proof. ◀

For the next reduction, we will utilize the lower bound on the norm of vectors in a so-called Graver basis of a matrix. For two vectors $y, x \in \mathbb{Z}^n$ we write $y \lhd x$ if for every $i \in [n]$ it holds that $y_i x_i \ge 0$ and $|y_i| \le |x_i|$. A non-zero vector $x \in \mathbb{Z}^n$ belongs to the Graver basis of $A \in \mathbb{Z}^{m \times n}$ if $Ax = 0$ and no other non-zero solution $Ay = 0$ satisfies $y \lhd x$. In other words, x encodes a sequence of columns of A, some possibly repeated or negated, that sums to 0 and none of its nontrivial subsequences sums to 0. The following lemma concerns the existence of vectors with a large ℓ_1-norm in a Graver basis of a certain matrix. We state it in the matrix-column interpretation.

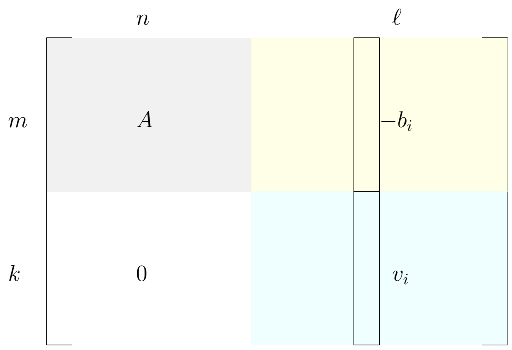


Figure 1 The matrix A' in Lemma 20.

▶ **Lemma 19** ([12, Thm. 9, Cor. 5]). *For every $k \in \mathbb{N}$ there is a sequence $(v_1, \dots, v_n)$ of vectors from $\{-1, 0, 1\}^k$ such that*

1. *$n = \Theta(2^k)$,*
2. *the vectors $v_1, \dots, v_n$ sum up to 0, and*
3. *no proper non-empty subsequence of $(v_1, \dots, v_n)$ sums up to 0.*

▶ **Lemma 20.** Monotone 0-1 ILP$[m]$ $\le_{PPT}$ 0-Sum 0-1 ILP$[m]$.

Proof. Consider an instance $Ax = b$ of Monotone 0-1 ILP$[m]$ with $A \in \{0,1\}^{m \times n}$. We can assume that A contains 1 in every column as otherwise such a column can be discarded. Let $v_1, \dots, v_\ell \in \{-1, 0, 1\}^k$ be the sequence of vectors from Lemma 19 with $\ell \ge n$ and $k = \mathcal{O}(\log n)$. Next, we can assume that $||b||_\infty \le n \le \ell$ as otherwise there can be no solution. We decompose b into a sum $b_1 + \cdots + b_\ell$ of vectors from $\{-1, 0, 1\}^m$, possibly using zero-vectors for padding. Now we construct a matrix $A' \in \{-1, 0, 1\}^{(m+k)\times(n+\ell)}$. The first n columns are given by the columns of A with 0 on the remaining k coordinates. The last ℓ columns are of the form $(-b_i, v_i)$ for $i \in [\ell]$. See Figure 1 for an illustration. The new parameter is $m + k$ which is $m + \mathcal{O}(\log n)$.

We claim that $Ax = b$ is feasible over boolean domain if and only if $A'y = 0$ admits a non-zero boolean solution y. Consider a solution x to $Ax = b$. We define y as x concatenated with vector 1^ℓ. In each of the first m rows we have $(A'y)_j = (Ax)_j - b_j = 0$. In the remaining k rows we have first n zero vectors followed by the sequence $v_1, \dots, v_\ell$ which sums up to 0 by construction. Hence $A'y = 0$ while $y \ne 0$.

Now consider the other direction and let $y \ne 0$ be a solution to $A'y = 0$. Let us decompose y as a concatenation of $y_1 \in \{0,1\}^n$ and $y_2 \in \{0,1\}^\ell$. First suppose that $y_2 = 0$. Then $y_1 \ne 0$ and $Ay_1 = 0$ but this is impossible since $A \in \{0,1\}^{m\times n}$ and, by assumption, every column of A contains 1. It remains to consider the case $y_2 \ne 0$. By inspecting the last k rows of A' we infer that the non-zero indices of y_2 correspond to a non-empty subsequence of $v_1, \dots, v_\ell$ summing up to 0. By construction, this is not possible for any proper subsequence of $(v_1, \dots, v_\ell)$ so we must have $y_2 = 1^\ell$. Hence $0 = A'y = A'y_1 + A'y_2 = A'y_1 + (-b, 0)$ and so $Ay_1 = b$. This concludes the proof of the reduction. ◀

We will now reduce from 0-Sum 0-1 ILP$[m]$ to 0-1 ILP$[m]$. The subtlety comes from the fact that in the latter problem we accept the solution $x = 0$ while in the first we do not. Observe that the reduction is easy when we can afford guessing a single column from a solution. For a matrix $A \in \mathbb{Z}^{m\times n}$ and $i \in [n]$ we denote by $A^i \in \mathbb{Z}^{m\times 1}$ the i-th column of A and by $A^{-i} \in \mathbb{Z}^{m\times(n-1)}$ the matrix obtained from A by removal of the i-th column.

▶ **Observation 21.** *An instance* $Ax = 0$ *of* 0-Sum 0-1 ILP$[m]$ *is solvable if and only if there is* $i \in [n]$ *such that the instance* $A^{-i}y = -A^i$ *of* 0-1 ILP$[m]$ *is solvable.*

▶ **Lemma 22.** 0-Sum 0-1 ILP$[m]$ $\leq_{NPPT}$ 0-1 ILP$[m]$.

Proof. Observation 21 enables us to solve 0-Sum 0-1 ILP$[m]$ in polynomial time when $\log n$ is large compared to m, by considering all $i \in [n]$ and solving the obtained 0-1 ILP$[m]$ instance. Hence we can again assume that $\log n \leq \text{poly}(m)$. In this case, Observation 21 can be interpreted as an NPPT that guesses $\text{poly}(m)$ bits to identify the index $i \in [n]$. ◀

▶ **Lemma 23.** 0-1 ILP$[m]$ $\leq_{NPPT}$ Monotone 0-1 ILP$[m]$.

Proof. We decompose the matrix A as $A^+ - A^-$ where A^+, A^- have entries from $\{0, 1\}$. Suppose that there exists a vector x satisfying $Ax = b$. We nondeterministically guess vectors b^+, b^- that satisfy $A^+x = b^+$, $A^-x = b^-$ and we check whether $b^+ - b^- = b$; if no then the guess is rejected. This requires $m \log n$ nondeterministic bits. We create an instance of Monotone 0-1 ILP$[m]$ with $2m$ constraints given as $A^+y = b^+$, $A^-y = b^-$. If we made a correct guess, then $y = x$ is a solution to the system above. On the other hand, if this system admits a solution y then $Ay = A^+y - A^-y = b^+ - b^- = b$ so y is also a solution to the original instance. ◀

▶ **Lemma 24.** Subset Sum$[\log t]$ $\equiv_{NPPT}$ Group-$\mathbb{Z}_q$ Subset Sum$[\log q]$.

Proof. For the reduction ($\leq$) consider $q = nt$ and leave t intact. We can assume that each input number belongs to $[1, t)$ hence the sum of every subset belongs to $[1, q)$ and so there is no difference in performing addition in $\mathbb{Z}$ or $\mathbb{Z}_q$.

Now we handle the reduction ($\geq$). Let S be the subset of numbers that sums up to t modulo q. Since each item belongs to $[0, q)$ their sum in $\mathbb{Z}$ is bounded by nq; let us denote this value as t'. We nondeterministically guess $t' \in [0, nq]$ and check whether $t' = t \mod m$. We consider an instance J of Subset Sum over $\mathbb{Z}$ with the unchanged items and the target t'. We have $\log t' \leq \log n + \log q$ what bounds the new parameter as well as the number of necessary nondeterministic bits. If the guess was correct then J will have a solution. Finally, a solution to J yields a solution to the original instance because $t' = t \mod q$. ◀

Using the presented lemmas, any two problems listed in Theorem 3 can be reduced to each other via NPPT.

4 Permutation Subset Sum

This section is devoted to the proof of Theorem 5. We will use an intermediate problem involving a computational model with ℓ binary counters, being a special case of bounded *Vector Addition System with States* (VASS) [68]. This can be also regarded as a counterpart of the intermediate problem used for establishing XNLP-hardness, which concerns cellular automata [16, 43].

For a sequence $\mathcal{F} = (f_1, \dots, f_n)$, $f_i \in \{O, R\}$ (optional/required), we say that a subsequence of $[n]$ is *$\mathcal{F}$-restricted* if it contains all the indices i with $f_i = R$. We say that a sequence of vectors $v_1, \dots, v_n \in \{-1, 0, 1\}^\ell$ forms a *0/1-run* if $v_1 + \dots + v_n = 0$ and for each $j \in [n]$ the partial sum $v_1 + \dots + v_j$ belongs to $\{0, 1\}^\ell$.

0-1 Counter Machine **Parameter:** ℓ
Input: Sequences $\mathcal{V} = (v_1, \dots, v_n)$, $v_i \in \{-1, 0, 1\}^\ell$, and $\mathcal{F} = (f_1, \dots, f_n)$, $f_i \in \{O, R\}$.
Question: Is there a subsequence $(i_1 < i_2 < \dots < i_r)$ of $[n]$ that is $\mathcal{F}$-restricted and such that $(v_{i_1}, v_{i_2}, \dots, v_{i_r})$ forms a 0/1-run?

Intuitively, a vector $v_i \in \{-1,0,1\}^\ell$ tells which of the ℓ counters should be increased or decreased. We must "execute" all the vector v_i for which $f_i = R$ plus some others so that the value of each counter is always kept within $\{0,1\}$.

We give a reduction from 3COLORING[PW] to 0-1 COUNTER MACHINE[ℓ].

3COLORING **Parameter:** PW
Input: An undirected graph G and a path decomposition of G of width at most PW
Question: Can we color $V(G)$ with 3 colors so that the endpoints of each edge are assigned different colors?

▶ **Lemma 25 (★).** 3COLORING[PW] $\leq_{PPT}$ 0-1 COUNTER MACHINE[ℓ].

In the proof, we assign each vertex a label from [PW + 1] so that the labels in each bag are distinct. We introduce a counter for each pair (label, color) and whenever a vertex is introduced in a bag, we make the machine increase one of the counters corresponding to its label. For each edge uv there is a bag containing both u, v; we then insert a suitable sequence of vectors so that running it is possible if and only if the labels of u, v have active counters in different colors. Finally, when a vertex is forgotten we deactivate the corresponding counter.

In order to encode the operations on counters as composition of permutations, we will employ the following algebraic construction. For $q \in \mathbb{N}$ consider an automorphism $\phi_1 \colon \mathbb{Z}_q^2 \to \mathbb{Z}_q^2$ given as $\phi_1((x,y)) = (y,x)$. Clearly $\phi_1 \circ \phi_1$ is identify, so there is a homomorphism $\phi\colon \mathbb{Z}_2 \to Aut(\mathbb{Z}_q^2)$ that assigns identity to $0 \in \mathbb{Z}_2$ and ϕ_1 to $1 \in \mathbb{Z}_2$. We define the group U_q as the outer semidirect product $\mathbb{Z}_q^2 \rtimes_\phi \mathbb{Z}_2$ (see Section 2). That is, the elements of U_q are $\{((x,y),z) \mid x,y \in \mathbb{Z}_q, z \in \mathbb{Z}_2\}$ and the group operation $\circ$ is given as

$$((x_1,y_1),z_1) \circ ((x_2,y_2),z_2) = \begin{cases} ((x_1 \oplus_q x_2),(y_1 \oplus_q y_2), z_1 \oplus_2 z_2) & \text{if } z_1 = 0 \\ ((x_1 \oplus_q y_2),(y_1 \oplus_q x_2), z_1 \oplus_2 z_2) & \text{if } z_1 = 1. \end{cases} \tag{1}$$

The z-coordinate works as addition modulo 2 whereas the element z_1 governs whether we add (x_2,y_2) or (y_2,x_2) modulo q on the (x,y)-coordinates. The neutral element is $((0,0),0)$. Note that U_q is non-commutative.

For $((x,y),z) \in U_q$ we define its *norm* as $x+y$. Consider a mapping $\Gamma\colon \{-1,0,1\} \to U_q$ given as $\Gamma(-1) = ((1,0),1)$, $\Gamma(0) = ((0,0),0)$, $\Gamma(1) = ((0,1),1)$.

▶ **Lemma 26.** *Let $b_1,\dots,b_n \in \{-1,0,1\}$ and $q > n$. Then $b_1,\dots,b_n$ forms a 0/1-run (in dimension $\ell = 1$) if and only if the group product $g = \Gamma(b_1) \circ \Gamma(b_2) \circ \cdots \circ \Gamma(b_n)$ in U_q is of the form $g = ((0,n'),0)$ for some $n' \in [n]$.*

Proof. Recall that $\Gamma(0)$ is the neutral element in U_q. Moreover, removing 0 from the sequence does not affect the property of being a 0/1-run, so we can assume that $b_i \in \{-1,1\}$ for each $i \in [n]$. Note that the inequality $q > n$ is preserved by this modification. This inequality is only needed to ensure that the addition never overflows modulo q.

Suppose now that $b_1,\dots,b_n$ is a 0/1-run. Then it comprises alternating 1s and -1s: $(1,-1,1,-1,\dots,1,-1)$. Hence the product $g = \Gamma(b_1) \circ \cdots \circ \Gamma(b_n)$ equals $(\Gamma(1) \circ \Gamma(-1))^{n/2}$. We have $((0,1),1) \circ ((1,0),1) = ((0,2),0)$ and so $g = ((0,n),0)$.

Now suppose that $b_1,\dots,b_n$ is not a 0/1-run. Then either $\sum_{i=1}^n b_i = 1$ or $\sum_{i=1}^j b_i \notin \{0,1\}$ for some $j \in [n]$. In the first scenario n is odd so g has 1 on the z-coordinate and so it is not in the form of $((0,n'),0)$. In the second scenario there are 3 cases: (a) $b_1 = -1$, (b) $(b_i,b_{i+1}) = (1,1)$ for some $i \in [n-1]$, or (c) $(b_i,b_{i+1}) = (-1,-1)$ for some $i \in [n-1]$.

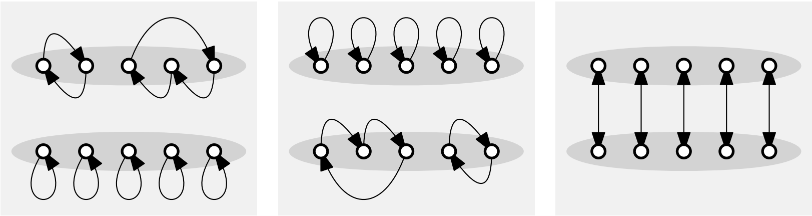


Figure 2 An illustration to Lemma 27. The three permutations are $\pi_1, \pi_0, \pi_z \in S_{10}$. The first one acts as g on the upper set and as identity on the lower set. In the second one these roles are swapped whereas π_z acts as symmetry between the two sets. The permutations $\widehat{\Gamma}(1), \widehat{\Gamma}(-1)$ in Lemma 28 are obtained as $\pi_0 \circ \pi_z$ and $\pi_1 \circ \pi_z$. Multiplying a sequence of permutations from $\{\widehat{\Gamma}(1), \widehat{\Gamma}(-1)\}$ yields a permutation acting as identity on the upper set if and only if the arguments are alternating 1s and -1s.

Case (a): $g = \Gamma(-1) \circ h = ((1,0),1) \circ h$ for some $h \in U_q$ of norm $\leq n-1$. Then g cannot have 0 at the x-coordinate because $n < q$ and the addition does not overflow.

Case (b): $\Gamma(1)^2 = ((0,1),1)^2 = ((0,1) \oplus_q (1,0), 1 \oplus_2 1) = ((1,1),0)$. For any $h_1, h_2 \in U_q$ of total norm $\leq n-1$ the product $h_1 \circ ((1,1),0) \circ h_2$ cannot have 0 at the x-coordinate.

Case (c): Analogous to (b) because again $\Gamma(-1)^2 = ((1,0),1)^2 = ((1,0) \oplus_q (0,1), 1 \oplus_2 1) = ((1,1),0)$. ◀

Next, we show how to embed the group U_q into a permutation group over a universe of small size. On an intuitive level, we need to implement two features: counting modulo q on both coordinates and a mechanism to swap the coordinates. To this end, we will partition the universe into two sets corresponding to the two coordinates. On each of them, we will use a permutation of order q to implement counting without interacting with the other set. Then we will employ a permutation being a bijection between the two sets, which will work as a switch. See Figure 2 for a visualization.

▶ **Lemma 27.** *For every $n \in \mathbb{N}$ there exist $q > n$ and $\hat{r} = \mathcal{O}(\log^3 n)$ for which there is a homomorphism $\chi \colon U_q \to S_{\hat{r}}$.*

Proof. By Lemma 13 we can find a permutation g of order $q > n$ in S_r for some $r = \mathcal{O}(\log^3 n)$. The subgroup of S_r generated by g is isomorphic to $\mathbb{Z}_q$. We will now consider the permutation group over the set $[r] \times \mathbb{Z}_2$, which is isomorphic to S_{2r}. Instead of writing χ explicitly, we will identify a subgroup of S_{2r} isomorphic to U_q.

Let π_z be the permutation given as $\pi_z(i,j) = (i, 1-j)$ for $(i,j) \in [r] \times \mathbb{Z}_2$, i.e., it switches the second coordinate. Let π_0 act as g on $[r] \times 0$ and as identify on $[r] \times 1$. Analogously, let π_1 act as g on $[r] \times 1$ and as identify on $[r] \times 0$. Let N be the subgroup of S_{2r} generated by π_0 and π_1; it is isomorphic to $\mathbb{Z}_q^2$ and each element of N is of the form (π_0^x, π_1^y) for some $x, y \in \mathbb{Z}_q$. Next, let H be the subgroup generated by π_z; it is isomorphic to $\mathbb{Z}_2$. Now consider a homomorphism $\phi \colon H \to Aut(N)$ given as conjugation $\phi_\pi(g) = \pi \circ g \circ \pi^{-1}$. In this special case, the semidirect product $N \rtimes_\phi H$ is isomorphic to the subgroup of S_{2r} generated by the elements of N and H (Lemma 14). On the other hand, ϕ_{π_z} maps $(\pi_0^x, \pi_1^y) \in N$ into (π_0^y, π_1^x) so this is exactly the same construction as used when defining U_q. We infer that U_q is isomorphic to a subgroup of S_{2r} and the corresponding homomorphism is given by the mapping of the generators: $\chi((0,1),0) = \pi_0$, $\chi((1,0),0) = \pi_1$, $\chi((0,0),1) = \pi_z$. ◀

Armed with such a homomorphism, we translate Lemma 26 to the language of permutations.

▶ **Lemma 28.** *For every $n \in \mathbb{N}$ there exists $r = \mathcal{O}(\log^3 n)$, a permutation $\pi \in S_r$ of order greater than n, and a mapping $\widehat{\Gamma}\colon \{-1,0,1\} \to S_r$ so that the following holds. A sequence $b_1, \dots, b_n \in \{-1,0,1\}$ is a 0/1-run if and only if the product $\widehat{\Gamma}(b_1) \circ \widehat{\Gamma}(b_2) \circ \dots \circ \widehat{\Gamma}(b_n)$ is of the form $\pi^{n'}$ for some $n' \in [n]$.*

Proof. Let $\chi\colon U_q \to S_r$ be the homomorphism from Lemma 27 for $q > n$ and $r = \mathcal{O}(\log^3 n)$. We define $\widehat{\Gamma}\colon \{-1,0,1\} \to S_r$ as $\widehat{\Gamma}(i) = \chi(\Gamma(i))$ using the mapping Γ from Lemma 26. Since χ is a homomorphism, the condition $\Gamma(b_1) \circ \dots \circ \Gamma(b_n) = ((0,n'),0)$ is equivalent to $\widehat{\Gamma}(b_1) \circ \dots \circ \widehat{\Gamma}(b_n) = \chi(((0,n'),0))$. We have $g = \chi(((0,n'),0))$ for some $n' \in [n]$ if and only if $g = \pi^{n'}$ for $\pi = \chi((0,1),0)$. The order of π is $q > n$, as requested. ◀

For a sequence of vectors from $\{0,1\}^\ell$ we can use a Cartesian product of ℓ permutation groups S_r to check the property of being a 0/1-run by inspecting the product of permutations from $S_{\ell r}$. This enables us to encode the problem with binary counters as GROUP-S_k SUBSET SUM$[k]$. We remark that we need nondeterminism to guess the target permutation. This boils down to guessing the number n' from Lemma 28 for each of ℓ coordinates.

Finally, Theorem 5 follows by combining Lemma 25 with Lemma 29.

▶ **Lemma 29 (★).** 0-1 COUNTER MACHINE$[\ell]$ $\le_{NPPT}$ GROUP-S_k SUBSET SUM$[k]$.

5 Conclusion

We have introduced the nondeterministic polynomial parameter transformation (NPPT) and used this concept to shed some light on the unresolved questions about short certificates for FPT problems. We believe that our work will give an impetus for further systematic study of certification complexity in various contexts.

The main question remains to decipher certification complexity of SUBSET SUM$[\log t]$. Even though SUBSET SUM enjoys a seemingly simple structure, some former breakthroughs required advanced techniques such as additive combinatorics [3, 26] or number theory [61]. Theorem 3 makes it now possible to analyze SUBSET SUM$[\log t]$ through the geometric lens using concepts such as lattice cones [41] or Graver bases [12, 13, 67].

Drucker et al. [39] suggested also to study k-DISJOINT PATHS and K-CYCLE in their regime of polynomial witness compression. Recall that the difference between that model and ours is that they ask for a *randomized* algorithm that *outputs* a solution. Observe that a polynomial certificate (or witness compression) for k-DISJOINT PATHS would entail an algorithm with running time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$ which seems currently out of reach [62]. What about a certificate of size $(k + \log n)^{\mathcal{O}(1)}$?

Interestingly, PLANAR k-DISJOINT PATHS does admit a polynomial certificate: if $k^2 \le \log n$ one can execute the known $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$-time algorithm [73] and otherwise one can guess the homology class of a solution (out of $n^{\mathcal{O}(k)} \le 2^{\mathcal{O}(k^3)}$) and then solve the problem in polynomial time [87]. Another interesting question is whether k-DISJOINT PATHS admits a certificate of size $(k + \log n)^{\mathcal{O}(1)}$ on acyclic digraphs. Note that we need to incorporate $(\log n)$ in the certificate size because the problem is W[1]-hard when parameterized by k [88]. The problem admits an $n^{\mathcal{O}(k)}$-time algorithm based on dynamic programming [48].

For K-CYCLE we cannot expect to rule out a polynomial certificate via a PPT from AND-3SAT$[k]$ because the problem admits a polynomial compression [90], a property unlikely to hold for AND-3SAT$[k]$ [47]. Is it possible to establish the certification hardness by NPPT (which does not preserve polynomial compression) or would such a reduction also lead to unexpected consequences?

A different question related to bounded nondeterminism is whether one can rule out a logspace algorithm for directed reachability (which is NL-complete) using only polylog(n) nondeterministic bits. Observe that relying on the analog of the assumption NP $\not\subseteq$ coNP/poly for NL would be pointless because NL = coNL by Immerman-Szelepcsényi Theorem. This direction bears some resemblance to the question whether directed reachability can be solved in polynomial time and polylogarithmic space, i.e., whether NL $\subseteq$ SC [1].

References

1 Scott Aaronson. The complexity zoo, 2005. URL: `https://cse.unl.edu/~cbourke/latex/ComplexityZoo.pdf`.

2 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND subset sum. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.ICALP.2020.4`.

3 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. `doi:10.1145/3450524`.

4 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of mathematics*, pages 781–793, 2004.

5 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory Comput.*, 10:297–339, 2014. `doi:10.4086/TOC.2014.V010A012`.

6 Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 450–457. ACM / IEEE Computer Society, 2002. `doi:10.1145/774572.774638`.

7 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

8 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense subset sum may be the hardest. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPICS.STACS.2016.13`.

9 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 57–67. SIAM, 2021. `doi:10.1137/1.9781611976496.6`.

10 Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 58–69. SIAM, 2019. `doi:10.1137/1.9781611975482.4`.

11 Richard Bellman. Dynamic programming. *Princeton University Press, Princeton, NJ, USA*, 1:3–25, 1958.

12 Sebastian Berndt, Matthias Mnich, and Tobias Stamm. New support size bounds and proximity bounds for integer linear programming. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 82–95. Springer, 2024. `doi:10.1007/978-3-031-52113-3_6`.

13 Yael Berstein and Shmuel Onn. The Graver complexity of integer programming. *Annals of Combinatorics*, 13:289–296, 2009.

14 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice: The tape model. *Information and Computation*, 254:59–83, 2017. doi:10.1016/J.IC.2017.03.001.

15 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.8.

16 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.

17 Hans L. Bodlaender, Isja Mannens, Jelle J. Oostveen, Sukanya Pandey, and Erik Jan van Leeuwen. The parameterised complexity of integer multicommodity flow. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.6.

18 Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017. doi:10.1145/3056461.

19 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

20 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/S00453-022-00928-W.

21 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1797–1815. SIAM, 2021. doi:10.1137/1.9781611976465.108.

22 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

23 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021. doi:10.3233/FAIA200990.

24 Jean Cardinal and John Iacono. Modular subset sum, dynamic strings, and zero-sum sets. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 45–56. SIAM, 2021. doi:10.1137/1.9781611976496.5.

25 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

26 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Approximating partition in near-linear time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 307–318. ACM, 2024. doi:10.1145/3618260.3649727.

27 Stephen Cook and Phuong Nguyen. *Logical foundations of proof complexity*, volume 11. Cambridge University Press Cambridge, 2010.

28 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

29 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

30 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

31 Stefan S. Dantchev, Barnaby Martin, and Stefan Szeider. Parameterized proof complexity. *Comput. Complex.*, 20(1):51–85, 2011. doi:10.1007/S00037-010-0001-1.

32 Evgeny Dantsin and Edward A. Hirsch. Satisfiability certificates verifiable in subexponential time. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 19–32, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21581-0_4.

33 Holger Dell. AND-compression of NP-complete problems: Streamlined proof and minor observations. *Algorithmica*, 75(2):403–423, 2016. doi:10.1007/S00453-015-0110-Y.

34 Antoine Deza, Asaf Levin, Syed Mohammad Meesum, and Shmuel Onn. Hypergraphic degree sequences are hard. *Bull. EATCS*, 127, 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/573/572.

35 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009. doi:10.1051/ITA/2009012.

36 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

37 Andrew Drucker. Nondeterministic direct product reductions and the success probability of SAT solvers. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 736–745, USA, 2013. IEEE Computer Society. doi:10.1109/FOCS.2013.84.

38 Andrew Drucker. Nondeterministic direct product reductions and the success probability of SAT solvers, 2013. URL: https://people.csail.mit.edu/andyd/success_prob_long.pdf.

39 Andrew Drucker, Jesper Nederlof, and Rahul Santhanam. Exponential Time Paradigms Through the Polynomial Time Lens. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.36.

40 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006. doi:10.3233/SAT190014.

41 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

42 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

43 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/S00453-014-9944-Y.

44 Paul Erdős. Graphs with prescribed degree of vertices. *Mat. Lapok.*, 11:264–274, 1960.

45 Michael R. Fellows and Frances A. Rosamond. Collaborating with Hans: Some remaining wonderments. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 7–17. Springer, 2020. doi:10.1007/978-3-030-42071-0_2.

46 Nick Fischer and Leo Wennmann. Minimizing tardy processing time on a single machine in near-linear time. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 64:1–64:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.64.

47 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

48 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

49 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991. doi:10.1137/0220072.

50 Carla Groenland, Gwenaël Joret, Wojciech Nadara, and Bartosz Walczak. Approximating pathwidth for graphs of small treewidth. *ACM Trans. Algorithms*, 19(2):16:1–16:19, 2023. doi:10.1145/3576044.

51 Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers*. Oxford university press, 1979.

52 Danny Harnik and Moni Naor. On the compressibility of NP instances and cryptographic applications. *SIAM Journal on Computing*, 39(5):1667–1713, 2010. doi:10.1137/060668092.

53 Danny Hermelin, Stefan Kratsch, Karolina Sołtys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/S00453-014-9910-8.

54 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010. doi:10.1007/978-3-642-13190-5_12.

55 Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptol.*, 9(4):199–216, 1996. doi:10.1007/BF00189260.

56 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

57 Bart M. P. Jansen, Shivesh Kumar Roy, and Michał Włodarczyk. On the hardness of compressing weights. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 64:1–64:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.MFCS.2021.64.

58 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ITCS.2019.43.

59 Ce Jin, Nikhil Vyas, and Ryan Williams. Fast low-space algorithms for subset sum. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1757–1776. SIAM, 2021. doi:10.1137/1.9781611976465.106.

60 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015. doi:10.1007/978-3-319-23219-5_15.

61 Daniel M. Kane. Unary subset-sum is in logspace. *arXiv*, abs/1012.1336, 2010. arXiv:1012.1336.

62 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. doi:10.1016/J.JCTB.2011.07.004.

63 Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003. doi:10.1016/S0022-0000(03)00006-0.

64 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ilps. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2947–2960. SIAM, 2023. doi:10.1137/1.9781611977554.CH112.

65 Johannes Köbler and Jochen Messner. Is the standard proof system for SAT p-optimal? In Sanjiv Kapoor and Sanjiva Prasad, editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FSTTCS 2000 New Delhi, India, December 13-15, 2000, Proceedings*, volume 1974 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2000. doi:10.1007/3-540-44450-5_29.

66 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

67 Taisei Kudo and Akimichi Takemura. A lower bound for the Graver complexity of the incidence matrix of a complete bipartite graph. *Journal of Combinatorics*, 3(4):695–708, 2013.

68 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Wegrzycki. Coverability in VASS Revisited: Improving Rackoff's Bound to Obtain Conditional Optimality. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 131:1–131:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.131.

69 Jeffrey C Lagarias and Andrew M Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, 1985. doi:10.1145/2455.2461.

70 Eugene L Lawler and J Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management science*, 16(1):77–84, 1969.

71 Francis Lazarus and Arnaud de Mesmay. Knots and 3-dimensional computational topology, 2017. URL: https://pagesperso.g-scop.grenoble-inp.fr/~lazarusf/Enseignement/compuTopo6.pdf.

72 Andrew Peter Lin. *Solving hard problems in election systems*. Rochester Institute of Technology, 2012. URL: https://repository.rit.edu/cgi/viewcontent.cgi?article=1332&context=theses.

73 Daniel Lokshtanov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1307–1316. ACM, 2020. doi:10.1145/3357713.3384250.

74 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.

75 Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE transactions on Information Theory*, 24(5):525–530, 1978. doi:10.1109/TIT.1978.1055927.

76 Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. A subquadratic approximation scheme for partition. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 70–88. SIAM, 2019. doi:10.1137/1.9781611975482.5.

77 Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a target interval to a few exact queries. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 718–727. Springer, 2012. doi:10.1007/978-3-642-32589-2_62.

78 Jesper Nederlof and Karol Wegrzycki. Improving Schroeppel and Shamir's algorithm for subset sum via orthogonal vectors. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.

79 Jean-Louis Nicolas. *On Landau's Function $g(n)$*, pages 228–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi:10.1007/978-3-642-60408-9_18.

80 Ramamohan Paturi and Pavel Pudlak. On the complexity of circuit satisfiability. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, pages 241–250, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806724.

81 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

82 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999. doi:10.1006/JAGM.1999.1034.

83 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and subset sum with small items. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.106.

84 Krzysztof Potepa. Faster deterministic modular subset sum. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 76:1–76:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.76.

85 Neil Robertson and Paul D Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/J.JCTB.2004.08.001.

86 Derek JS Robinson. *An introduction to abstract algebra.* Walter de Gruyter, 2003.

87 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994. doi:10.1137/S0097539792224061.

88 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discret. Math.*, 24(1):146–157, 2010. doi:10.1137/070697781.

89 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-boolean optimization by implicit hitting sets. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 51:1–51:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.51.

90 Magnus Wahlström. Abusing the Tutte matrix: An algebraic instance compression for the K-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.STACS.2013.341.

91 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

92 Michał Włodarczyk. Does subset sum admit short proofs? *arXiv*, abs/2409.03526, 2024. arXiv:2409.03526.

Approximation Algorithms for Cumulative Vehicle Routing with Stochastic Demands

Jingyang Zhao
University of Electronic Science and Technology of China, Chengdu, China

Mingyu Xiao[1]
University of Electronic Science and Technology of China, Chengdu, China

Abstract

In the Cumulative Vehicle Routing Problem (Cu-VRP), we need to find a feasible itinerary for a capacitated vehicle located at the depot to satisfy customers' demand, as in the well-known Vehicle Routing Problem (VRP), but the goal is to minimize the cumulative cost of the vehicle, which is based on the vehicle's load throughout the itinerary. If the demand of each customer is unknown until the vehicle visits it, the problem is called Cu-VRP with Stochastic Demands (Cu-VRPSD). In this paper, we propose a randomized 3.456-approximation algorithm for Cu-VRPSD, improving the best-known approximation ratio of 6 (Discret. Appl. Math. 2020). Since VRP with Stochastic Demands (VRPSD) is a special case of Cu-VRPSD, as a corollary, we also obtain a randomized 3.25-approximation algorithm for VRPSD, improving the best-known approximation ratio of 3.5 (Oper. Res. 2012). At last, we give a randomized 3.194-approximation algorithm for Cu-VRP, improving the best-known approximation ratio of 4 (Oper. Res. Lett. 2013).

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Cumulative Vehicle Routing, Stochastic Demands, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.59

Funding The work is supported by the National Natural Science Foundation of China, under the grants 62372095, 62172077, and 62350710215.

1 Introduction

In the well-known Vehicle Routing Problem (VRP) [7], we are given an undirected complete graph $G = (V, E)$ with $V = \{v_0, v_1, \ldots, v_n\}$, where v_0 denotes the depot, and the other n vertices denote n customers. Moreover, there is a weight function w on the edges representing the length of edges, which satisfies the triangle inequality, and a demand vector $d = (d_1, ..., d_n)$ implying that each customer v_i has a demand of d_i. The objective is to determine an *itinerary* for a vehicle with a capacity of Q, starting from and ending at the depot, that fulfills every customer's demand while minimizing the total weight of the edges in the itinerary.

In the Cumulative Vehicle Routing Problem (Cu-VRP) [18, 19], the goal is also to find an itinerary for the vehicle, but with the objective of minimizing the *cumulative cost* of the itinerary. Here, the cumulative cost for the vehicle traveling from u to v carrying a load of $x_{uv} \leq Q$ units of goods is defined as $a \cdot w(u, v) + b \cdot x_{uv} \cdot w(u, v)$, where $a, b \in \mathbb{R}_{\geq 0}$ are given parameters. Cu-VRP captures the fuel consumption in transportation and logistics, as fuel consumption depends on both the weight of the empty vehicle and the weight of the goods being carried by the vehicle [10]. Since fuel consumption can account for as much as 60% of a vehicle's operational costs [23], Cu-VRP has been studied extensively through both experimental algorithms [27, 25, 9, 13, 22] and approximation algorithms [10, 11, 12]. A recent survey of Cu-VRP can be found in [6].

[1] Corresponding author

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 59; pp. 59:1–59:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In VRP with Stochastic Demands (VRPSD) [2], the demand of each customer is represented by an independent random variable with a known distribution, and its value is unknown until the vehicle visits the customer. The goal is to design a policy such that the expected weight of the itinerary is minimized. Early surveys on this topic can be found in [14, 3]. Cu-VRP with Stochastic Demands (Cu-VRPSD) was proposed in [11], and similarly, the goal is to design a policy such that the expected cumulative cost of the itinerary is minimized. In VRPSD, we can fully load the vehicle before it leaves the depot. However, in Cu-VRPSD, due to the fact that both higher and lower loads can lead to higher cumulative costs, we need to carefully consider how the vehicle is loaded. This property makes Cu-VRPSD both more challenging and more interesting compared to VRPSD.

In each of the above problems, the *splittable* (resp., *unsplittable*) variant requires that the demand of each customer can be satisfied partially within the vehicle's visits (resp., must be satisfied entirely in one of the vehicle's visits).

In this paper, we consider approximation algorithms for the unsplittable variants. Note that the unsplittable variants are more difficult. For example, unsplittable VRP generalizes the bin packing problem even on a line shape graph [26], and thus cannot be approximated with an approximation ratio of less than 1.5 unless P=NP. For Cu-VRP, an algorithm is called a ρ-approximation algorithm if it can output a solution with a cumulative cost of at most $\rho \cdot \mathrm{OPT}$ in polynomial time, where OPT is the cumulative cost of the optimal solution. For Cu-VRPSD, an algorithm is called a ρ-approximation algorithm if it can employ a policy to get a solution with an expected cumulative cost of at most $\rho \cdot \mathrm{OPT}$ in polynomial time, where OPT is the cumulative cost of the minimum expected cumulative cost solution obtained by the optimal policy.

Let α denote the approximation ratio of the metric Traveling Salesman Problem (TSP). It is well-known that $\alpha \leq 1.5$ [5, 24], which is slightly improved to $\alpha \leq 1.5 - 10^{-36}$ [20, 21]. The ratio α for TSP will be frequently used in VRP related problems.

For VRP, there is an $(\alpha + 1)$-approximation algorithm for the splittable case [17], and an $(\alpha + 2)$-approximation algorithm for the unsplittable case [1]. Blauth *et al.* [4] improved the ratio to $\alpha + 1 - \varepsilon$ for the splittable case, and Friggstad *et al.* [8] improved the ratio to $\alpha + 1 + \ln 2 - \varepsilon'$ for the unsplittable case using the LP rounding method, where ε and ε' are small positive constants related to α. Notably, Friggstad *et al.* [8] also gave a combinatorial $(\alpha + 1.75 - \varepsilon')$-approximation algorithm for the unsplittable case. For VRPSD, there is a randomized $(\alpha + 1 + o(1))$-approximation algorithm for the splittable case, and a randomized $(\alpha + Q)$-approximation algorithm for the unsplittable case. Gupta *et al.* [16] improved the ratios to $\alpha + 1$ and $\alpha + 2$, respectively.

For Cu-VRP, Gaur *et al.* [10] proposed a $(1 + \frac{4\alpha}{\sqrt{4\alpha^2+24\alpha+4}-2\alpha})$-approximation algorithm for the splittable case, and a $(1+\frac{4\alpha}{\sqrt{4\alpha^2+24\alpha+4}-(2\alpha+2)})$-approximation algorithm for the unsplittable case. For Cu-VRPSD, Gaur *et al.* [12] gave a randomized $\max\{1+\frac{3}{2}\alpha, 3\}$-approximation algorithm for the splittable case, and a randomized $\max\{2 + \frac{3}{2}\alpha, 6\}$-approximation algorithm for the unsplittable case.

1.1 Our results

In this paper, we focus on the unsplittable cases of Cu-VRPSD, VRPSD, and Cu-VRP, and design improved approximation algorithms for them.

The main idea of the most recent algorithms [16, 12] is as follows. First, we find an α-approximate TSP tour and then the vehicle satisfies customers in the order they appear on the TSP tour. Once the load is less than the serving customer's demand the vehicle goes back to the depot to reload. Our algorithms will also use an α-approximate TSP tour and visit the customers in the order according to the TSP tour. However, we do not strictly satisfy the customers in the order. To reduce the cumulative cost, our vehicle may skip customers with large demands when visiting customers according to the TSP tour (but record their demands) and satisfy them after completing the TSP tour.

Based on the above idea, we propose two novel algorithms for Cu-VRPSD, denoted as $ALG.1(\lambda, \delta)$ and $ALG.2(\lambda, \delta)$.

- In $ALG.1(\lambda, \delta)$, the vehicle will skip customers in $\{v_i \mid d_i > \lambda \cdot Q\}$ and then satisfy each of them by using a single tour;
- In $ALG.2(\lambda, \delta)$, the vehicle will skip customers in $\{v_i \mid d_i > \delta \cdot Q\}$ and then satisfy them either by using a single tour for each or by calling an algorithm for weighted set cover.

Furthermore, in our algorithms, we set upper and lower bounds of the load of the vehicle when traveling along the TSP tour: the load is at least $\delta \cdot Q$ and less than $\lambda \cdot Q$ for some parameters δ and λ. The lower bound can be regarded as the *backup* goods that the vehicle carries. The idea of carrying some backup goods was inspired by a partition algorithm for the TSP tour used for unsplittable VRP [8]. We will show that this approach can reduce the potential cumulative cost caused by visiting customers with demands at most $\delta \cdot Q$ at the expense of increasing the cumulative cost of the vehicle when traveling the TSP tour. So, we need to balance the setting of δ, e.g., we may set $\delta = 0$ when a/b is small.

We will prove that $ALG.1(\lambda, \delta)$ can be used to obtain a randomized algorithm with an expected approximation ratio of $10/3$ for $a/b \leq 0.375$ and 3.456 for $0.375 < a/b \leq 1.444$, and by using both $ALG.1(\lambda, \delta)$ and $ALG.2(\lambda, \delta)$, we can get a randomized 3.456-approximation algorithm for $a/b > 1.444$. Hence, we get a randomized 3.456-approximation algorithm for Cu-VRPSD.

Note that Cu-VRPSD reduces to VRPSD when $b = 0$, and this corresponds to $a/b = \infty$. As a corollary, for VRPSD, we also obtain a randomized 3.25-approximation algorithm using the randomized 3.456-approximation algorithm for Cu-VRPSD with $a/b > 1.444$.

For Cu-VRP, we also give two algorithms, denoted as $ALG.3(\lambda, \delta)$ and $ALG.4(\lambda)$. Since the demands of customers are known in advance, in $ALG.3(\lambda, \delta)$, we first obtain a set of tours by applying the randomized rounding method to the LP of weighted set cover, and then satisfy the remaining customers by calling $ALG.1(\lambda, \delta)$; in $ALG.4(\lambda)$, we directly call $ALG.1(\lambda, 0)$. In the tours obtained by calling $ALG.1(\lambda, \delta)$ and $ALG.1(\lambda, 0)$, the load of the vehicle may be greater than the delivered units of goods. So, we also adapt a pre-optimization step to ensure that the load of the vehicle equals the delivered units of goods.

We will show that $ALG.3(\lambda, \delta)$ can be used to obtain a randomized 3.194-approximation algorithm with a running time of $n^{O(\frac{1}{\min\{a/b,1\}})}$ and thus it only works for $a/b > \gamma_0$, where $\gamma_0 > 0$ is any fixed constant, and $ALG.4(\lambda, \delta)$ can be used to obtain a randomized 3.163-approximation algorithm for $a/b < 0.428$. Hence, we get a randomized 3.194-approximation algorithm for Cu-VRP.

A summary of our results under $\alpha = 1.5$ can be found in Table 1. Although our algorithms are simple and neat, the analysis is technically involved. Some parts also need careful calculation. To avoid distraction from our main discussions and also due to the limited space, the proofs of lemmas and theorems marked with "*" are omitted.

2 Notations

In Cu-VRP, we use $G = (V, E)$ to denote the input complete graph, where $V = \{v_0, \ldots, v_n\}$. There is a non-negative weight function $w : E \to \mathbb{R}_{\geq 0}$ on the edges, where $w(u, v)$ denotes the length of edge $uv \in E$. We assume that w is a *metric*, i.e., it is symmetric and satisfies the

Table 1 A summary of the previous approximation ratios and our approximation ratios.

	Previous Results	Our Results
Cu-VRPSD	6 [12]	**3.456**
VRPSD	3.5 [16]	**3.25**
Cu-VRP	4 [10]	**3.194**

triangle inequality. Let $V' := V \setminus \{v_0\}$. There is also a demand vector $d = (d_1, ..., d_n) \in \mathbb{R}^{V'}_{[0,Q]}$, where $Q \in \mathbb{R}_{>0}$ is the capacity of the vehicle, and each customer v_i has a required demand $d_i \in [0, Q]$. We let $l_i := w(v_0, v_i)$ and $[i] := \{1, 2, ..., n\}$.

In Cu-VRPSD, the demand of each customer v_i is represented by an independent random variable $\chi_i \in [0, Q]$, where the distribution of χ_i is usually assumed to be known in advance [2]. Let $\chi = (\chi_1, ..., \chi_n)$, where we assume that χ_i is not identically zero, as v_i can be ignored in such a case. Consequently, any feasible policy must visit every customer at least once [16].

For any random variable L, we use $L \sim U[l, r)$ to indicate that L is uniformly distributed over the interval $[l, r)$, where $l < r$.

A *tour* $T = v_0 v_1 \dots v_i v_0$ is a directed simple cycle, which always contains the depot v_0. We use $E(T)$ to denote the set of edges on T, and $V'(T)$ to denote the set of customers on T. Assume that the vehicle carries a load of x_{eT} units of goods when traveling along $e \in E(T)$. The cumulative cost of T is

$$Cu(T) := a \cdot \sum_{e \in E(T)} w(e) + b \cdot \sum_{e \in E(T)} x_{eT} \cdot w(e),$$

where $w(T) := \sum_{e \in E(T)} w(e)$ is called the weight of T, $Cu_1(T) := a \cdot \sum_{e \in E(T)} w(e)$ is called the vehicle cost of T, and $Cu_2(T) := b \cdot \sum_{e \in E(T)} x_{eT} \cdot w(e)$ is called the cargo cost of T. An itinerary $\mathcal{T}$ is a set of tours. A *TSP tour* is an undirected cycle that includes all customers and the depot exactly once. The weight of the minimum weight TSP tour is denoted by τ.

2.1 Problem Definitions

▶ **Definition 1** (**Cu-VRPSD**). *Given a complete graph $G = (V, E)$, a metric weight function w, a vehicle capacity $Q \in \mathbb{R}_{>0}$, a random demand variable vector $\chi = (\chi_1, ..., \chi_n)$, and two parameters $a, b \in \mathbb{R}_{\geq 0}$, we need to design a policy to find a feasible itinerary $\mathcal{T}$ such that*

- *the vehicle carries at most Q units of goods on each tour $T \in \mathcal{T}$,*
- *the vehicle delivers goods to customers only in $V'(T)$ on each tour $T \in \mathcal{T}$,*
- *the sum of the delivered demand over all tours for each $v_i \in V'$ equals the demand of v_i,*

and $\mathbb{E}[Cu(\mathcal{T})]$ is minimized.

Note that the demand of each customer is unknown until the vehicle visits it. In Cu-VRP, we have $\chi = d$, where d is known in advance. We assume that the deliveries are unsplittable: each customer may be included in multiple tours, but its demand must be satisfied entirely within exactly one of those tours. Moreover, by scaling each customer's demand χ_i to χ_i/Q and adjusting the parameter b to $b \cdot Q$, without loss of generality, we assume that $Q = 1$.

2.2 The Lower bounds

To analyze approximation algorithms, we recall the following lower bound for Cu-VRPSD.

▶ **Lemma 2** ([12]). *For unsplittable Cu-VRPSD, it holds that $\mathbb{E}[Cu(\mathcal{T}^*)] \geq a \cdot \max\{\tau, \sum_{i \in [n]} 2 \cdot \mathbb{E}[\chi_i] \cdot l_i\} + b \cdot \sum_{i \in [n]} \mathbb{E}[\chi_i] \cdot l_i$.*

When $a = 1$ and $b = 0$, the lower bound in Lemma 2 becomes $\max\{\tau, \sum_{i\in[n]} 2\cdot\mathbb{E}[\chi_i]\cdot l_i\}$, and it was used in analyzing approximation algorithms for VRPSD in [16]. To analyze our algorithms, we use a stronger lower bound that was implicitly used in the proof of Lemma 2.

▶ **Lemma 3** ([12])**.** *For unsplittable Cu-VRPSD with any demand realization vector $d \in \mathbb{R}^{V'}_{[0,1]}$, it holds that $\mathbb{E}[Cu(\mathcal{T}^*) \mid \chi = d] \geq LB := a\cdot\max\{\tau,\ \eta\} + b\cdot 0.5\cdot\eta$, where $\eta := \sum_{i\in[n]} 2\cdot d_i\cdot l_i$.*

Lemma 3 is stronger than Lemma 2 since it holds that $\mathbb{E}[\max\{X, Y\}] \geq \max\{\mathbb{E}[X], \mathbb{E}[Y]\}$ for any random variables X and Y by the property of the maximum function.

▶ **Lemma 4.** *An algorithm is a ρ-approximation algorithm for Cu-VRPSD if, for any possible demand realization vector $d \in \mathbb{R}^{V'}_{[0,1]}$, the algorithm conditioned on $\chi = d$ outputs a solution $\mathcal{T}$ with a cumulative cost of $\mathbb{E}[Cu(\mathcal{T}) \mid \chi = d] \leq \rho\cdot LB$.*

Proof. Since it holds that $\mathbb{E}[Cu(\mathcal{T}) \mid \chi = d] \leq \rho\cdot LB \leq \rho\cdot\mathbb{E}[Cu(\mathcal{T}^*) \mid \chi = d]$ for any possible demand realization vector d, we can get that $\mathbb{E}[Cu(\mathcal{T}) \mid \chi] \leq \rho\cdot\mathbb{E}[Cu(\mathcal{T}^*) \mid \chi]$. Therefore, we have $\mathbb{E}[Cu(\mathcal{T})] = \mathbb{E}[\mathbb{E}[Cu(\mathcal{T}) \mid \chi]] \leq \rho\cdot\mathbb{E}[\mathbb{E}[Cu(\mathcal{T}^*) \mid \chi]] = \rho\cdot\mathbb{E}[Cu(\mathcal{T}^*)]$. ◀

Hence, we may frequently analyze our algorithms conditioned on $\chi = d$, where $d \in \mathbb{R}^{V'}_{[0,1]}$ is any possible demand realization. For the sake of analysis, we let $\gamma := a/b$, and $\sigma := \gamma/\eta$. Note that $b = 0$ corresponds to the case where $\gamma = \infty$, which turns out to be easier, as will be shown in Theorem 14. We also define

$$\int_l^r x^t dF(x) := \frac{\sum_{v_i\in V': l<d_i\leq r} 2\cdot d_i^t\cdot l_i}{\sum_{v_i\in V'} 2\cdot d_i\cdot l_i}, \qquad \text{where} \quad t \in \{0, 1, 2\}. \tag{1}$$

Note that $\int_0^1 x dF(x) = 1$. Moreover, for any $0 \leq l \leq r$, we have

$$l\cdot\int_l^r x^{t-1}dF(x) < \int_l^r x^t dF(x) \leq r\cdot\int_l^r x^{t-1}dF(x). \tag{2}$$

3 Two Algorithms for Cu-VRPSD

3.1 The first algorithm

In this section, we will introduce our first algorithm, denoted as $ALG.1(\lambda, \delta)$, which can be used to get a 10/3-approximation algorithm for Cu-VRPSD with any $\gamma \in (0, 0.375]$ and a 3.456-approximation algorithm for Cu-VRPSD with any $\gamma \in [0.375, 1.444]$. Here, $\lambda \in (0, 1]$ and $\delta \in [0, \lambda/2]$ are parameters that will be defined later.

Firstly, $ALG.1(\lambda, \delta)$ computes an α-approximate TSP tour T^*, which will be oriented in either clockwise or counterclockwise direction. Assume that $T^* = v_0v_1\dots v_nv_0$ by renumbering the customers following the orientation. Then, the vehicle in $ALG.1(\lambda, \delta)$ tries to satisfy the customers in the order of $v_1\dots v_n$ as they appear on T^*, where the parameters λ and δ ensures that the load of the vehicle during its travel on each edge of T^* is at least δ and less than λ. Moreover, among its load, the δ units of goods are regarded as *backup* goods, and the other units of goods are regarded as *normal* goods. Specifically, if the vehicle carries L_{i-1} demand of normal goods during its travel from v_{i-1} to v_i, we have $0 \leq L_{i-1} < \lambda - \delta$ for each $i \in [n+1]$. We say that the vehicle carries $S_{i-1} = (L_{i-1}, \delta)$ units of goods to indicate that it carries L_{i-1} demand of normal goods and δ demand of backup goods. We require that $0 < \lambda \leq 1$ and $0 \leq \delta \leq \lambda - \delta$, i.e., $0 \leq \delta \leq \lambda/2$. When serving a customer, the main strategy is to prioritize using the normal goods first and then consider using the backup goods if the normal goods are insufficient. Conditioned on $\chi = d$, the details are as follows.

Initially, we load the vehicle with $S_0 = (L_0, \delta)$ units of goods at the depot, where $L_0 \sim U[0, \lambda - \delta)$. When the vehicle is about to serve v_i, we assume that it carries $S_{i-1} = (L_{i-1}, \delta)$ units of goods, where $0 \leq L_{i-1} < \lambda - \delta$. Then, we have the following three cases.

Case 1: $d_i \leq L_{i-1}$. In this case, the vehicle directly delivers $(d_i, 0)$ units of goods for v_i, and then goes to the next customer. Hence, we have $S_i = (L_i, \delta)$, where $L_i \coloneqq L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda-\delta} \rceil \cdot (\lambda - \delta) - d_i = L_{i-1} - d_i$ since $d_i \leq L_{i-1} < \lambda - \delta$.

Case 2: $L_{i-1} < d_i \leq L_{i-1} + \delta$. The vehicle delivers $(L_{i-1}, d_i - L_{i-1})$ units of goods for v_i, goes to the depot to reload $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda-\delta} \rceil \cdot (\lambda - \delta) - d_i, d_i - L_{i-1})$ units of goods, and then goes to the next customer. Hence, we have $S_i = (L_i, \delta)$, where $L_i \coloneqq L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda-\delta} \rceil \cdot (\lambda - \delta) - d_i = L_{i-1} + (\lambda - \delta) - d_i$ since $0 < d_i - L_{i-1} \leq \delta \leq \lambda - \delta$.

Case 3: $L_{i-1} + \delta < d_i \leq 1$. In this case, we must have $d_i > \delta$.

Case 3.1: $\delta < d_i \leq \lambda$. The vehicle goes to the depot to reload $(d_i - L_{i-1} - \delta, 0)$ units of goods, goes to satisfy v_i, then goes to the depot to reload $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda-\delta} \rceil \cdot (\lambda - \delta) - d_i, \delta)$ units of goods, goes to customer v_i again (for the sake of analysis), and then goes to the next customer. Hence, we have $S_i = (L_i, \delta)$, where $L_i \coloneqq L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda-\delta} \rceil \cdot (\lambda - \delta) - d_i$.

Case 3.2: $\lambda < d_i \leq 1$. Since $L_{i-1} < \lambda - \delta$, we must have $L_{i-1} + \delta < d_i$. Instead of satisfying v_i by returning to the depot to reload as in Case 3.1, the vehicle records its demand, skips it, and goes to the next customer. Hence, we have $S_i = (L_i, \delta)$, where $L_i \coloneqq L_{i-1}$.

After trying to satisfy all customers using the above strategy, due to Case 3.2, there may be still a set of *unsatisfied* customers $\{v_i \mid d_i > \lambda\}$. Then, for each unsatisfied customer v_i, since its demand has been recorded, the vehicle will load exactly d_i units of goods at the depot, go to satisfy v_i, and then return to the depot.

The details of $ALG.1(\lambda, \delta)$ is shown in Algorithm 1.

Compared to the previous strategy in [12], there are two main differences. The first is that we specially handle each customer v_i with $d_i > \lambda$ in Case 3.2. Note that if we satisfy v_i as the method in Case 3.1, since $L_{i-1} + \delta < \lambda$ (we will prove it in Lemma 6), the vehicle must incur two visits to the depot, which will cost too much. The second is that we ensure that the vehicle always carries δ demand of backup goods when traveling along the TSP tour. The advantage is that each customer v_i with $d_i \leq \delta$ incurs at most one visit to the depot while if $\delta = 0$ every customer v_i with $d_i \leq \lambda$ has the potential to incur two visits to the depot. However, since $\delta > 0$ clearly increases the cumulative cost of the vehicle when it travels along the TSP tour, we need to carefully set the value of δ.

Although we require that $\lambda > 0$ in $ALG.1(\lambda, \delta)$, it can be extended to the case of $\lambda = 0$. In this scenario, the vehicle simply travels along the TSP tour with an empty carry to record each customer's demand, and then satisfies each customer within a single tour, as described in Case 3.2. Interestingly, if $a = 0$, this algorithm becomes an exact algorithm for unsplittable Cu-VRPSD, as the cumulative cost is $b \cdot \sum_{i \in [n]} d_i \cdot l_i$, which matches the lower bound LB in Lemma 4. The running time can reach $O(n)$ since all TSP tours have the same performance. However, it may be useless for $a > 0$. Hence, we consider $\lambda > 0$ in the following.

▶ **Lemma 5.** *Unsplittable Cu-VRPSD with $a = 0$ can be solved in $O(n)$ time.*

3.1.1 The analysis

Note that $ALG.1(\lambda, \delta)$ carries $L_0 \sim U[0, \lambda - \delta)$ demand of normal goods initially. Next, we analyze the expected cumulative cost of $\mathcal{T}$ conditioned on $\chi = d$, i.e., $\mathbb{E}[Cu(\mathcal{T}) \mid \chi = d]$.

Algorithm 1 An algorithm for unsplittable Cu-VRPSD ($ALG.1(\lambda, \delta)$).

Input: An instance of unsplittable Cu-VRPSD, and two parameters $\lambda \in (0, 1]$ and $\delta \in [0, \lambda/2]$.
Output: A feasible solution $\mathcal{T}$ to unsplittable Cu-VRPSD.

```
1:  Obtain an $\alpha$-approximate TSP tour $T^*$ using an $\alpha$-approximation algorithm for metric TSP, orient $T^*$ in either clockwise or counterclockwise direction, and denote $T^* = v_0v_1v_2 \ldots v_nv_0$ by renumbering the customers following the direction.
2:  Load the vehicle with $S_0 := (L_0, \delta)$ units of goods, including $L_0$ demand of normal goods and $\delta$ demand of backup goods, where $L_0 \sim U[0, \lambda - \delta)$.
3:  Initialize $i := 1$ and $V^* := \emptyset$.
4:  while $i \le n$ do
5:      Go to customer $v_i$;
6:      if $d_i \le L_{i-1}$ then
7:          Deliver $(d_i, 0)$ units of goods to $v_i$, and update $S_i := (L_i, \delta)$, where $L_i := L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i = L_{i-1} - d_i$;
8:      else if $L_{i-1} < d_i \le L_{i-1} + \delta$ then
9:          Deliver $(L_{i-1}, d_i - L_{i-1})$ units of goods to $v_i$, goes to the depot, load the vehicle with $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, d_i - L_{i-1})$ units of goods, and update $S_i := (L_i, \delta)$, where $L_i := L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i = L_{i-1} + (\lambda - \delta) - d_i$;
10:     else if $L_i + \delta < d_i \le 1$ then
11:         if $\delta < d_i \le \lambda$ then
12:             Return to the depot, load the vehicle with $(d_i - L_{i-1} - \delta, 0)$ units of goods, go to customer $v_i$, and deliver $(d_i - \delta, \delta)$ units of goods to $v_i$.
13:             Return to the depot, load the vehicle with $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, \delta)$ units of goods, go to customer $v_i$, and update $S_i := (L_i, \delta)$, where $L_i := L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i$;
                ▷ The vehicle returns to $v_i$ for the sake of analysis; however, it could directly proceed to $v_{i+1}$.
14:         else if $\lambda < d_i \le 1$ then
15:             Record $v_i$'s demand, and update $V^* := V^* \cup \{v_i\}$ and $S_i := (L_i, \delta)$, where $L_i := L_{i-1}$;
16:         end if
17:     end if
18:     $i := i + 1$.
19: end while
20: Go to the depot.
21: for $v_i \in V^*$ do
22:     Load the vehicle with $d_i$ units of goods, go to customer $v_i$, and deliver $d_i$ units of goods to $v_i$;
23:     Go to the depot.
24: end for
```

In $ALG.1(\lambda, \delta)$, the vehicle carries $S_{i-1} = (L_{i-1}, \delta)$ units of goods when traveling along the edge $v_{i-1}v_{i \bmod (n+1)}$ of the TSP tour T^*. For each $i \in [n]$, we let $h_i := 1$ if $d_i \le \lambda$, and $h_i := 0$ otherwise. We have the following lemma.

▶ **Lemma 6.** *For any $i \in [n+1]$, it holds $L_{i-1} = L_0 + \lceil \frac{\sum_{j=1}^{i-1} h_j \cdot d_j - L_0}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - \sum_{j=1}^{i-1} h_j \cdot d_j$, and moreover, $L_{i-1} \sim U[0, \lambda - \delta)$, conditioned on $\chi = d$.*

Proof. Since $L_0 \sim U[0, \lambda - \delta)$, the lemma holds for $i = 1$. Assume that the equality holds for $i = i' \ge 1$, i.e., $L_{i'-1} = L_0 + \lceil \frac{\sum_{j=1}^{i'-1} h_j \cdot d_j - L_0}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - \sum_{j=1}^{i'-1} h_j \cdot d_j$. Note that we have $0 \le L_{i'-1} < \lambda - \delta$. Next, we consider $L_{i'}$.

Case 1: $d_{i'} \le \lambda$. We have $h_{i'} = 1$. By Lines 7, 9, and 13, we have $L_{i'} = L_{i'-1} + \lceil \frac{d_{i'} - L_{i'-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_{i'}$. Hence, we have $L_{i'} \ge 0$ and $L_{i'} < \lambda - \delta$. Therefore, we have $L_0 + \lceil \frac{\sum_{j=1}^{i'-1} h_j \cdot d_j - L_0}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - \sum_{j=1}^{i'} h_j \cdot d_j + \lceil \frac{d_{i'} - L_{i'-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) \ge 0$ and

$L_0+\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'}h_j\cdot d_j+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta)<\lambda-\delta$. Alternatively, we have $\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil-1<\frac{\sum_{j=1}^{i'}h_j\cdot d_j-L_0}{\lambda-\delta}\leq\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil$, and hence $\lceil\frac{\sum_{j=1}^{i'}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil=\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil$. Therefore, we have $L_{i'}=L_{i'-1}+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-d_{i'}=L_0+\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'}h_j\cdot d_j+\lceil\frac{d_{i'}-L_{i'-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta)=L_0+\lceil\frac{\sum_{j=1}^{i'}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'}h_j\cdot d_j$.

Case 2: $d_i>\lambda$. We have $h_i=0$, and hence $\sum_{j=1}^{i'-1}h_j\cdot d_j=\sum_{j=1}^{i'}h_j\cdot d_j$. By Line 15, we have $L_{i'}=L_{i'-1}=L_0+\lceil\frac{\sum_{j=1}^{i'-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'-1}h_j\cdot d_j=L_0+\lceil\frac{\sum_{j=1}^{i'}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'}h_j\cdot d_j$.

In both cases, we have $L_{i'}=L_0+\lceil\frac{\sum_{j=1}^{i'}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i'}h_j\cdot d_j$. By induction, the equality holds for any $i\in[n+1]$.

For any $i\in[n+1]$, we have $L_{i-1}=L_0+\lceil\frac{\sum_{j=1}^{i-1}h_j\cdot d_j-L_0}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-\sum_{j=1}^{i-1}h_j\cdot d_j$. Assume that $(\sum_{j=1}^{i-1}h_j\cdot d_j)\bmod(\lambda-\delta)=L'$, which is fixed conditioned on $\chi=d$. We have $L_{i-1}=\lambda-\delta+L_0-L'\in[\lambda-\delta-L',\lambda-\delta)$ when $L_0\in[0,L')$, and $L_{i-1}=L_0-L'\in[0,\lambda-\delta-L')$ when $L_0\in[L',\lambda-\delta)$. The relationship between L_0 and L_{i-1} is bijective. Since $L_0\sim U[0,\lambda-\delta)$, we can also get $L_{i-1}\sim U[0,\lambda-\delta)$, conditioned on $\chi=d$. ◀

Lemma 6 also implies that $0\leq L_{i-1}<\lambda-\delta$ for any $i\in[n+1]$.

▶ **Lemma 7.** *In ALG.*$1(\lambda,\delta)$*, the expected cumulative cost conditioned on* $\chi=d$ *during the vehicle's travel from* v_{i-1} *to* v_i *is* $a\cdot w(v_{i-1},v_i)+b\cdot\frac{\lambda+\delta}{2}\cdot w(v_{i-1},v_i)$.

Proof. By Lemma 6, the vehicle carries (L_{i-1},δ) units of goods during the vehicle's travel from v_{i-1} to v_i, where $L_{i-1}\sim U[0,\lambda-\delta)$. So, $\mathbb{E}[L_{i-1}+\delta\mid\chi=d]=\int_0^{\lambda-\delta}\frac{x+\delta}{\lambda-\delta}dx=\frac{\lambda+\delta}{2}$. Hence, the expected cumulative cost of the vehicle's travel from v_{i-1} to v_i conditioned on $\chi=d$ is $a\cdot w(v_{i-1},v_i)+b\cdot\mathbb{E}[L_{i-1}+\delta\mid\chi=d]\cdot w(v_{i-1},v_i)=a\cdot w(v_{i-1},v_i)+b\cdot\frac{\lambda+\delta}{2}\cdot w(v_{i-1},v_i)$. ◀

By Line 9 in *ALG.*$1(\lambda,\delta)$, if the vehicle visits v_i carrying $S_{i-1}=(L_{i-1},\delta)$ units of goods, where $L_{i-1}<d_i\leq L_{i-1}+\delta$, it will first satisfy v_i, then proceed to the depot to reload some units of goods, and finally return to the place of v_i. We refer to this process as *one additional visit to* v_0.

By Lines 12 and 13 in *ALG.*$1(\lambda,\delta)$, if the vehicle visits v_i with $d_i\leq\lambda$ carrying $S_{i-1}=(L_{i-1},\delta)$ units of goods, where $L_{i-1}+\delta<d_i$, it will first go to the depot to reload some units of goods, then go to the place of v_i to satisfy v_i, proceed to the depot to reload some units of goods, and finally return to the place of v_i again. We refer to this process as *two additional visits to* v_0.

When the vehicle is about to serve v_i with $d_i\leq\lambda$, it may incur one additional visit or two additional visits to v_0, resulting in some cumulative cost. For each customer v_i with $d_i>\lambda$, By Line 22, the vehicle satisfies v_i using a single tour, which will also be regarded as one additional visit to v_0 for the sake of presentation. Next, we analyze the expected cumulative cost conditioned on $\chi=d$ due to the possible additional visit(s) for each customer v_i.

▶ **Lemma 8.** *Conditioned on* $\chi=d$*, when serving each customer* v_i *in ALG.*$1(\lambda,\delta)$*, the expected cumulative cost of the vehicle due to the possible additional visit(s) to* v_0 *is*

- $a\cdot\frac{2d_i}{\lambda-\delta}\cdot l_i+b\cdot\frac{(\lambda+\delta)\cdot d_i-d_i^2}{\lambda-\delta}\cdot l_i$ *if* $d_i\leq\delta$*;*

- $a \cdot \frac{4d_i-2\delta}{\lambda-\delta} \cdot l_i + b \cdot \frac{d_i^2+(\lambda-\delta)\cdot d_i}{\lambda-\delta} \cdot l_i$ *if* $\delta < d_i \leq \lambda - \delta$*;*
- $a \cdot \frac{2d_i+2\lambda-4\delta}{\lambda-\delta} \cdot l_i + b \cdot \frac{2d_i^2-(\lambda+\delta)\cdot d_i+\lambda^2-\delta^2}{\lambda-\delta} \cdot l_i$ *if* $\lambda - \delta < d_i \leq \lambda$*;*
- $a \cdot 2 \cdot l_i + b \cdot d_i \cdot l_i$ *if* $\lambda < d_i \leq 1$*.*

Proof. If $d_i > \lambda$, By Lines 15 and 22, the vehicle incurs one additional visit to v_0, where the vehicle carries d_i units of goods from v_0 to v_i and 0 units of goods from v_i to v_0. So, the expected cumulative cost is $a \cdot 2 \cdot l_i + b \cdot d_i \cdot l_i$. Next, we consider $d_i \leq \lambda$.

By Lemma 6, the vehicle carries (L_{i-1}, δ) units of goods when traveling along $v_{i-1}v_i$ in $ALG.1(\lambda, \delta)$, and it holds that $L_{i-1} \sim U[0, \lambda - \delta)$. Hence, the vehicle incurs one additional visits to v_0 with a probability of $\Pr[d_i - \delta \leq L_{i-1} = x < d_i]$, and incurs two additional visits to v_0 with a probability of $\Pr[0 \leq L_{i-1} = x < d_i - \delta]$. We consider the following three cases.

Case 1: $d_i \leq \delta$. The vehicle incurs at most one additional visit to v_0. If the vehicle incurs one additional visit to v_0, By Line 9, the vehicle carries $(0, \delta - (d_i - L_{i-1}))$ units of goods from v_i to v_0, and $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, \delta) = (L_{i-1} + (\lambda - \delta) - d_i, \delta)$ units of goods from v_0 to v_i. So, the cumulative cost is $a \cdot 2 \cdot l_i + b \cdot (\delta - (d_i - L_{i-1}) + L_{i-1} + (\lambda - \delta) - d_i + \delta) \cdot l_i = a \cdot 2 \cdot l_i + b \cdot (2L_{i-1} - 2d_i + \lambda + \delta) \cdot l_i$. Since $L_{i-1} \sim U[0, \lambda - \delta)$ and $d_i \leq \delta \leq \lambda - \delta$, the expected cumulative cost is

$$\begin{aligned}
&\int_0^{\min\{d_i, \lambda-\delta\}} \frac{a \cdot 2 \cdot l_i + b \cdot (2x - 2d_i + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx \\
&= \int_0^{d_i} \frac{a \cdot 2 \cdot l_i + b \cdot (2x - 2d_i + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx = a \cdot \frac{2d_i}{\lambda - \delta} \cdot l_i + b \cdot \frac{(\lambda + \delta) \cdot d_i - d_i^2}{\lambda - \delta} \cdot l_i.
\end{aligned}$$

Case 2: $\delta < d_i \leq \lambda - \delta$. The vehicle incurs at most two additional visits to v_0. Similarly, if the vehicle incurs one additional visit to v_0, By Line 9, the vehicle carries $(0, \delta - (d_i - L_{i-1}))$ units of goods from v_i to v_0, and $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, \delta) = (L_{i-1} + (\lambda - \delta) - d_i, \delta)$ units of goods from v_0 to v_i, and the cumulative cost is $a \cdot 2 \cdot l_i + b \cdot (\delta - (d_i - L_{i-1}) + L_{i-1} + (\lambda - \delta) - d_i + \delta) \cdot l_i = a \cdot 2 \cdot l_i + b \cdot (2L_{i-1} - 2d_i + \lambda + \delta) \cdot l_i$. If the vehicle incurs two additional visits to v_0, By Lines 12 and 13, the vehicle carries (L_{i-1}, δ) units of goods from v_i to v_0, $(d_i - \delta, \delta)$ units of goods from v_0 to v_i, $(0, 0)$ units of goods from v_i to v_0, and $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, \delta) = (L_{i-1} + (\lambda - \delta) - d_i, \delta)$ units of goods from v_0 to v_i, and the cumulative cost is $a \cdot 4 \cdot l_i + b \cdot (L_{i-1} + \delta + d_i + 0 + L_{i-1} + (\lambda - \delta) - d_i + \delta) \cdot l_i = a \cdot 4 \cdot l_i + b \cdot (2L_{i-1} + \lambda + \delta) \cdot l_i$. Hence, the expected cumulative cost is

$$\begin{aligned}
&\int_{d_i - \delta}^{\min\{d_i, \lambda-\delta\}} \frac{a \cdot 2 \cdot l_i + b \cdot (2x - 2d_i + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx + \int_0^{d_i - \delta} \frac{a \cdot 4 \cdot l_i + b \cdot (2x + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx \\
&= \int_{d_i - \delta}^{d_i} \frac{a \cdot 2 \cdot l_i + b \cdot (2x - 2d_i + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx + \int_0^{d_i - \delta} \frac{a \cdot 4 \cdot l_i + b \cdot (2x + \lambda + \delta) \cdot l_i}{\lambda - \delta} dx \\
&= \frac{a \cdot 2 \cdot \delta \cdot l_i + b \cdot \lambda \cdot \delta \cdot l_i}{\lambda - \delta} + \frac{a \cdot 4 \cdot (d_i - \delta) \cdot l_i + b \cdot (d_i^2 + (\lambda - \delta) \cdot d_i - \delta \cdot \lambda) \cdot l_i}{\lambda - \delta} \\
&= a \cdot \frac{4d_i - 2\delta}{\lambda - \delta} \cdot l_i + b \cdot \frac{d_i^2 + (\lambda - \delta) \cdot d_i}{\lambda - \delta} \cdot l_i.
\end{aligned}$$

Case 3: $\lambda - \delta < d_i \leq \lambda$. The vehicle incurs at most two additional visits to v_0. If the vehicle incurs one additional visit to v_0, By Line 9, the vehicle carries $(0, \delta - (d_i - L_{i-1}))$ units of goods from v_i to v_0, and $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, \delta)$ units of goods from v_0 to v_i, and the cumulative cost is $a \cdot 2 \cdot l_i + b \cdot (\delta - (d_i - L_{i-1}) + L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i + \delta) \cdot l_i = a \cdot 2 \cdot l_i + b \cdot (2L_{i-1} - 2d_i + 2\delta + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta)) \cdot l_i$. If the vehicle incurs two additional visits to v_0, By Lines 12 and 13, the vehicle carries (L_{i-1}, δ) units of goods from v_i to v_0, $(d_i - \delta, \delta)$ units of goods from v_0 to v_i, $(0, 0)$ units of goods from v_i to v_0, and

$(L_{i-1}+\lceil\frac{d_i-L_{i-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-d_i,\delta)$ units of goods from v_0 to v_i, and the cumulative cost is $a\cdot 4\cdot l_i+b\cdot(L_{i-1}+\delta+d_i+0+L_{i-1}+\lceil\frac{d_i-L_{i-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta)-d_i+\delta)\cdot l_i = a\cdot 4\cdot l_i+b\cdot(2L_{i-1}+2\delta+\lceil\frac{d_i-L_{i-1}}{\lambda-\delta}\rceil\cdot(\lambda-\delta))\cdot l_i$. Hence, the expected cumulative cost is

$$\begin{aligned}
&\int_{d_i-\delta}^{\min\{d_i,\lambda-\delta\}}\frac{a\cdot 2\cdot l_i+b\cdot(2x-2d_i+2\delta+\lceil\frac{d_i-x}{\lambda-\delta}\rceil\cdot(\lambda-\delta))\cdot l_i}{\lambda-\delta}dx\\
&+\int_0^{d_i-\delta}\frac{a\cdot 4\cdot l_i+b\cdot(2x+2\delta+\lceil\frac{d_i-x}{\lambda-\delta}\rceil\cdot(\lambda-\delta))\cdot l_i}{\lambda-\delta}dx\\
=&\int_{d_i-\delta}^{\lambda-\delta}\frac{a\cdot 2\cdot l_i+b\cdot(2x-2d_i+2\delta+1\cdot(\lambda-\delta))\cdot l_i}{\lambda-\delta}dx\\
&+\int_0^{d_i+\delta-\lambda}\frac{a\cdot 4\cdot l_i+b\cdot(2x+2\delta+2\cdot(\lambda-\delta))\cdot l_i}{\lambda-\delta}dx\\
&+\int_{d_i+\delta-\lambda}^{d_i-\delta}\frac{a\cdot 4\cdot l_i+b\cdot(2x+2\delta+1\cdot(\lambda-\delta))\cdot l_i}{\lambda-\delta}dx\\
=&\frac{a\cdot 2\cdot(\lambda-d_i)\cdot l_i+b\cdot(d_i^2+(\delta-3\lambda)\cdot d_i+(2\lambda^2-\lambda\cdot\delta))}{\lambda-\delta}\\
&+\frac{a\cdot 4\cdot(d_i+\delta-\lambda)\cdot l_i+b\cdot(d_i^2+2\delta\cdot d_i+\delta^2-\lambda^2)}{\lambda-\delta}\\
&+\frac{a\cdot 4\cdot(\lambda-2\delta)\cdot l_i+b\cdot((2\lambda-4\delta)\cdot d_i+\delta\cdot\lambda-2\delta^2)}{\lambda-\delta}\\
=&\,a\cdot\frac{2d_i+2\lambda-4\delta}{\lambda-\delta}\cdot l_i+b\cdot\frac{2d_i^2-(\lambda+\delta)\cdot d_i+\lambda^2-\delta^2}{\lambda-\delta}\cdot l_i,
\end{aligned}$$

where the first equality follows from that $\lceil\frac{d_i-x}{\lambda-\delta}\rceil=1$ if $d_i+\delta-\lambda\leq x\leq\lambda-\delta$ and $\lceil\frac{d_i-x}{\lambda-\delta}\rceil=2$ if $0\leq x<d_i+\delta-\lambda$ since in our setting $\delta\leq\lambda-\delta$ and in this case $\lambda-\delta<d_i\leq\lambda$. ◀

▶ **Theorem 9 (*).** *For Cu-VRPSD with any $\lambda\in(0,1]$ and $\delta\in[0,\lambda/2]$, conditioned on $\chi=d$, $ALG.1(\lambda,\delta)$ generates a solution $\mathcal{T}$ with an expected cumulative cost of*

$$\frac{A+B}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}\cdot LB,$$

where $A:=\gamma\cdot(\alpha\cdot\sigma+\int_0^{\delta}\frac{x}{\lambda-\delta}dF(x)+\int_{\delta}^{\lambda-\delta}\frac{2x-\delta}{\lambda-\delta}dF(x)+\int_{\lambda-\delta}^{\lambda}\frac{x+\lambda-2\delta}{\lambda-\delta}dF(x)+\int_{\lambda}^{1}1dF(x))$, and $B:=(\frac{\lambda+\delta}{2}\cdot\alpha\cdot\sigma+\int_0^{\delta}\frac{(\lambda+\delta)x-x^2}{2(\lambda-\delta)}dF(x)+\int_{\delta}^{\lambda-\delta}\frac{x^2+(\lambda-\delta)x}{2(\lambda-\delta)}dF(x)+\int_{\lambda-\delta}^{\lambda}\frac{2x^2-(\lambda+\delta)\cdot x+\lambda^2-\delta^2}{2(\lambda-\delta)}dF(x)+\int_{\lambda}^{1}\frac{x}{2}dF(x))$.

3.1.2 The application

Next, we use $ALG.1(\lambda,\delta)$ to deign approximation algorithms for $a,b>0$, i.e., $\gamma>0$.

When the vehicle traveling along each edge of the TSP tour in $ALG.1(\lambda,\delta)$, it always carries at least δ units of goods in total, resulting in a large cumulative cost for the case where γ is small. Hence, intuitively, if γ is small, we simply set $\delta=0$.

If we call $ALG.1(\lambda,\delta)$ with $\delta=0$ and λ being a unique value, in the worst case we may have $\int_0^{\lambda}x^2dF(x)=\int_0^{\lambda}\lambda\cdot xdF(x)$, i.e., almost all customers v_i with $l_i>0$ and $d_i\leq\lambda$ have a demand of $d_i=\lambda$. Consequently, we may get $\int_0^{\theta\cdot\lambda}xdF(x)=0$ for any fixed $\theta\in(0,1)$, as will be shown in Lemma 10, and then $ALG.1(\theta\cdot\lambda,\delta)$ with $\delta=0$ and some $\theta\in(0,1)$ may generate a better solution. This suggests the approximation algorithm for Cu-VRPSD shown in Algorithm 2, denoted as $APPROX.1(\lambda,\theta,p)$.

Algorithm 2 An approximation algorithm for unsplittable Cu-VRPSD ($APPROX.1(\lambda, \theta, p)$).

```
Input: An instance of unsplittable Cu-VRPSD, and three parameters $\lambda \in (0,1]$, $\theta \in (0,1)$ and $p \in (0,1)$.
Output: A feasible solution to Cu-VRPSD.
1: Call $ALG.1(\lambda, 0)$ with a probability of $p$ and call $ALG.1(\theta \cdot \lambda, 0)$ with a probability of $1-p$.
```

Then, our goal is to find (λ, θ, p) minimizing the approximation ratio of $APPROX.1(\lambda, \theta, p)$.

▶ **Lemma 10.** *For any $\theta \in (0,1)$, we have $\int_0^{\theta\cdot\lambda} xdF(x) \leq \frac{1}{\lambda-\theta\cdot\lambda}(\int_0^\lambda \lambda\cdot xdF(x) - \int_0^\lambda x^2dF(x))$.*

Proof. By (1) and (2), we have $\int_0^\lambda x^2dF(x) = \int_0^{\theta\cdot\lambda} x^2dF(x) + \int_{\theta\cdot\lambda}^\lambda x^2dF(x) \leq \theta\cdot\lambda\cdot \int_0^{\theta\cdot\lambda} xdF(x) + \lambda\cdot\int_{\theta\cdot\lambda}^\lambda xdF(x) = \lambda\cdot\int_0^\lambda xdF(x) - (\lambda-\theta\cdot\lambda)\cdot\int_0^{\theta\cdot\lambda} xdF(x)$. Hence, we have $\int_0^{\theta\cdot\lambda} xdF(x) \leq \frac{1}{\lambda-\theta\cdot\lambda}(\int_0^\lambda \lambda\cdot xdF(x) - \int_0^\lambda x^2dF(x))$. ◀

▶ **Theorem 11.** *For unsplittable Cu-VRPSD, we can find (λ, θ, p) such that the approximation ratio of $APPROX.1(\lambda, \theta, p)$ is bounded by $10/3$ for any $\gamma \in (0, 0.375]$ and 3.456 for any $\gamma \in (0.375, 1.444]$.*

Proof. If we call $ALG.1(\lambda, \delta)$ with $\delta = 0$ and λ being a unique value, one may check that a good choice for λ is $\min\{1, 4\gamma/\alpha\}$. For the sake of analysis, we directly set $\lambda = \min\{1, 4\gamma/\alpha\}$.

If $\int_0^\lambda xdF(x) = 0$, we can get $\int_0^\lambda x^2dF(x) \leq \lambda\cdot\int_0^\lambda xdF(x) = 0$. Hence, we define $\mu := 0$ if $\int_0^\lambda xdF(x) = 0$, and $\mu := \frac{\int_0^\lambda x^2dF(x)}{\int_0^\lambda xdF(x)}$ otherwise.

By Lemma 4 and Theorem 9, the approximation ratio of $ALG.1(\lambda, 0)$ is at most

$$\begin{aligned}
&\max_{\sigma\geq 0} \frac{\gamma\cdot\left(\alpha\cdot\sigma + \int_0^\lambda \frac{2x}{\lambda}dF(x) + \int_\lambda^1 1dF(x)\right) + \left(\frac{\lambda}{2}\cdot\alpha\cdot\sigma + \int_0^\lambda \frac{x^2+\lambda\cdot x}{2\lambda}dF(x) + \int_\lambda^1 \frac{x}{2}dF(x)\right)}{\gamma\cdot\max\{\sigma,\ 1\} + 0.5}\\
&= \max_{\sigma\geq 0} \frac{\gamma\cdot\left(\alpha\cdot\sigma + \frac{2}{\lambda}\cdot\int_0^\lambda xdF(x) + \int_\lambda^1 1dF(x)\right) + \left(\frac{\lambda}{2}\cdot\alpha\cdot\sigma + \frac{1+\mu/\lambda}{2}\cdot\int_0^\lambda xdF(x) + \frac{1}{2}\cdot\int_\lambda^1 xdF(x)\right)}{\gamma\cdot\max\{\sigma,\ 1\} + 0.5}\\
&\leq \max_{\sigma\geq 0} \frac{\gamma\cdot\left(\alpha\cdot\sigma + \frac{2}{\lambda}\cdot\int_0^\lambda xdF(x) + \frac{1}{\lambda}\cdot\int_\lambda^1 xdF(x)\right) + \left(\frac{\lambda}{2}\cdot\alpha\cdot\sigma + \frac{1+\mu/\lambda}{2}\cdot\int_0^\lambda xdF(x) + \frac{1}{2}\cdot\int_\lambda^1 xdF(x)\right)}{\gamma\cdot\max\{\sigma,\ 1\} + 0.5}\\
&= \max_{\sigma\geq 0} \frac{\gamma\cdot\left(\alpha\cdot\sigma + \frac{1}{\lambda}\cdot\int_0^\lambda xdF(x) + \frac{1}{\lambda}\right) + \left(\frac{\lambda}{2}\cdot\alpha\cdot\sigma + \frac{\mu/\lambda}{2}\cdot\int_0^\lambda xdF(x) + \frac{1}{2}\right)}{\gamma\cdot\max\{\sigma,\ 1\} + 0.5}\\
&\leq \max_{\sigma\geq 0} \frac{\gamma\cdot(\alpha\cdot\sigma + \frac{2}{\lambda}) + \left(\frac{\lambda}{2}\cdot\alpha\cdot\sigma + \frac{\mu/\lambda+1}{2}\right)}{\gamma\cdot\max\{\sigma,\ 1\} + 0.5},
\end{aligned}$$

where the first equality follows from the definition of μ, the second equality from $\int_0^1 xdF(x) = 1$ by (1), the first inequality from $\int_\lambda^1 1dF(x) \leq \frac{1}{\lambda}\cdot\int_\lambda^1 xdF(x)$ by (2), and the second inequality from $\int_0^\lambda xdF(x) \leq \int_0^1 xdF(x) = 1$.

Since $\int_0^\lambda xdF(x) \leq \int_0^1 xdF(x) = 1$, by Lemma 10, we have

$$\begin{aligned}
\int_0^{\theta\cdot\lambda} xdF(x) &\leq \frac{1}{\lambda-\theta\cdot\lambda}\cdot\left(\int_0^\lambda \lambda\cdot xdF(x) - \int_0^\lambda x^2dF(x)\right)\\
&= \frac{\lambda-\mu}{\lambda-\theta\cdot\lambda}\cdot\int_0^\lambda xdF(x) \leq \frac{\lambda-\mu}{\lambda-\theta\cdot\lambda} \qquad (3)
\end{aligned}$$

Similarly, the approximation ratio of $ALG.1(\theta\cdot\lambda, 0)$ is at most

$$\begin{aligned}
&\max_{\sigma\geq 0}\frac{\gamma\cdot\left(\alpha\cdot\sigma+\int_0^{\theta\cdot\lambda}\frac{2x}{\theta\cdot\lambda}dF(x)+\int_{\theta\cdot\lambda}^1 1dF(x)\right)+\left(\frac{\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+\int_0^{\theta\cdot\lambda}\frac{x^2+\theta\cdot\lambda\cdot x}{2\cdot\theta\cdot\lambda}dF(x)+\int_{\theta\cdot\lambda}^1\frac{x}{2}dF(x)\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}\\
&\leq\max_{\sigma\geq 0}\frac{\gamma\cdot\left(\alpha\cdot\sigma+\frac{2}{\theta\cdot\lambda}\cdot\int_0^{\theta\cdot\lambda}xdF(x)+\frac{1}{\theta\cdot\lambda}\cdot\int_{\theta\cdot\lambda}^1 xdF(x)\right)+\left(\frac{\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+\int_0^{\theta\cdot\lambda}xdF(x)+\frac{1}{2}\cdot\int_{\theta\cdot\lambda}^1 xdF(x)\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}\\
&=\max_{\sigma\geq 0}\frac{\gamma\cdot\left(\alpha\cdot\sigma+\frac{1}{\theta\cdot\lambda}\cdot\int_0^{\theta\cdot\lambda}xdF(x)+\frac{1}{\theta\cdot\lambda}\right)+\left(\frac{\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+\frac{1}{2}\cdot\int_0^{\theta\cdot\lambda}xdF(x)+\frac{1}{2}\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}\\
&\leq\max_{\sigma\geq 0}\frac{\gamma\cdot\left(\alpha\cdot\sigma+\frac{1}{\theta\cdot\lambda}\cdot\frac{\lambda-\mu}{\lambda-\theta\cdot\lambda}+\frac{1}{\theta\cdot\lambda}\right)+\left(\frac{\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+\frac{1}{2}\cdot\frac{\lambda-\mu}{\lambda-\theta\cdot\lambda}+\frac{1}{2}\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}\\
&=\max_{\sigma\geq 0}\frac{\gamma\cdot\left(\alpha\cdot\sigma+\frac{1}{\theta\cdot\lambda}\cdot\frac{2\lambda-\mu-\theta\cdot\lambda}{\lambda-\theta\cdot\lambda}\right)+\left(\frac{\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+\frac{1}{2}\cdot\frac{2\lambda-\mu-\theta\cdot\lambda}{\lambda-\theta\cdot\lambda}\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5},
\end{aligned}$$

where the first inequality follows from (2), the second inequality from (3), and the first equality from $\int_0^1 xdF(x)=1$ by (1).

Recall that in $APPROX.1(\lambda,\theta,p)$ we call $ALG.1(\lambda,0)$ (resp., $ALG.1(\theta\cdot\lambda,0)$) with a probability of p (resp., $1-p$). Hence, to erase the items related to μ in the numerators of the approximation ratios of $ALG.1(\lambda,0)$ and $ALG.1(\theta\cdot\lambda,0)$, we need to set p such that $p\cdot\frac{1}{2}\cdot\frac{1}{\lambda}+(1-p)\cdot(\gamma\cdot\frac{1}{\theta\cdot\lambda}\cdot\frac{-1}{\lambda-\theta\cdot\lambda}+\frac{1}{2}\cdot\frac{-1}{\lambda-\theta\cdot\lambda})=0$. Then, we can get $p=\frac{\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}{\frac{1}{2\lambda}+\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}$. Clearly, we have $p\in[0,1]$. Hence, the approximation ratio is $\max_{\sigma\geq 0}R(\sigma)$, where

$$R(\sigma):=\frac{\gamma\cdot\left(\alpha\cdot\sigma+p\cdot\frac{2}{\lambda}+(1-p)\cdot\frac{1}{\theta\cdot\lambda}\cdot\frac{2\lambda-\theta\cdot\lambda}{\lambda-\theta\cdot\lambda}\right)+\left(\frac{p\cdot\lambda+(1-p)\cdot\theta\cdot\lambda}{2}\cdot\alpha\cdot\sigma+p\cdot\frac{1}{2}+(1-p)\cdot\frac{1}{2}\cdot\frac{2\lambda-\theta\cdot\lambda}{\lambda-\theta\cdot\lambda}\right)}{\gamma\cdot\max\{\sigma,\ 1\}+0.5}.$$

It is easy to check $\max_{\sigma\geq 0}R(\sigma)=\max_{\sigma\geq 1}R(\sigma)$. Moreover, since the function $\frac{a'x+b'}{c'x+d'}$ with $x\geq 1$ and $a',b',c',d'>0$ attains the maximum value only if $x=1$ or $x=\infty$, we know that the approximation ratio is bounded by $\max\{R(1),\ R(\infty)\}$. Recall that $\lambda=\min\{1,4\gamma/\alpha\}$ and $p=\frac{\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}{\frac{1}{2\lambda}+\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}$. Assume $\alpha=1.5$, and then we have $\alpha/4=0.375$. By calculation, we have the following results.

- When $\gamma\in(0,0.375]$, setting $\theta=0.5$, we have $\max\{R(1),\ R(\infty)\}\equiv 10/3$;
- When $\gamma\in[0.375,1.444]$, setting $\theta=0.6677$, we have $\max\{R(1),\ R(\infty)\}\leq 3.456$.

The result for $\gamma\in(0,0.375]$ may be surprising. We give the details of its proof.

▷ Claim 12. When $\gamma\in(0,0.375]$, setting $\theta=0.5$, we have $\max\{R(1),\ R(\infty)\}\equiv 10/3$.

Proof. Note that $\lambda=\min\{1,4\gamma/\alpha\}=4\gamma/\alpha$ and $\alpha=1.5$. Setting $\theta=0.5$, we can get $p=\frac{\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}{\frac{1}{2\lambda}+\frac{1}{2(\lambda-\theta\cdot\lambda)}+\frac{\gamma}{\theta\cdot\lambda(\lambda-\theta\cdot\lambda)}}=\frac{5}{6}$. Hence, under $\sigma\geq 1$, we have

$$R(\sigma)=\frac{3/2\cdot\gamma\cdot\sigma+5/8+3/8+11/6\cdot\gamma\cdot\sigma+2/3}{\gamma\cdot\sigma+0.5}=\frac{10}{3}\cdot\frac{\gamma\cdot\sigma+0.5}{\gamma\cdot\sigma+0.5}=\frac{10}{3}.$$

Hence, we have $\max\{R(1),\ R(\infty)\}\equiv 10/3$. ◁

This finishes the proof. ◀

We mention that the approximation ratio of $APPROX.1$ may achieve $\alpha+2=3.5$ when $\gamma=\infty$. Hence, it can not improve the current best approximation algorithm for VRPSD [16]. Additionally, a more careful design than $APPROX.1$ could yield improved approximations; however, the optimal design remains unknown.

3.2 The second algorithm

In this section, we will introduce our second algorithm, denoted as $ALG.2(\lambda, \delta)$, which can be used to get a 3.456-approximation algorithm for Cu-VRPSD with any $\gamma \in (1.444, \infty)$, and an $(\alpha + 1.75 = 3.25)$-approximation algorithm for VRPSD. Here, we may require $\lambda \in (0, 1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$.

$ALG.2(\lambda, \delta)$ is based on $ALG.1(\lambda, \delta)$. The vehicle will skip customers v_i with $d_i > \lambda$ when it travels along the TSP tour in $ALG.1(\lambda, \delta)$, and then satisfy each of them using a single tour at last. In $ALG.2(\lambda, \delta)$, the main difference is that the vehicle will skip customers v_i with $d_i > \delta$, and at last use the better method from either satisfying each of them using a single tour or solving the weighted $(1 - \delta)/\delta$-set cover problem as shown below.

Given a *feasible* set of unsatisfied customers S such that the total demand of all customers in S is at most 1, we know that the number of customers $|S|$ is at most $(1 - \delta)/\delta$ since each unsatisfied customer v_i has a demand of $d_i > \delta$. Then, we can optimally compute a tour with a minimum cumulative cost $Cu(S)$ for all customers in S in $O(|S|!)$ time. There are at most $n^{O(1/\delta)}$ number of feasible sets since $|S| \leq (1 - \delta)/\delta$. Therefore, to satisfy customers v_i with $d_i > \delta$, we can get an instance of weighted $(1 - \delta)/\delta$-set cover by taking each unsatisfied customer as an element, and each feasible set S of unsatisfied customers as a set with a weight of $Cu(S)$ in polynomial time. By calling a ρ-approximation algorithm for weighted $(1 - \delta)/\delta$-set cover [15], we can get a set of tours satisfying all customers v_i with $d_i > \delta$.

According to the two methods, there are two set of tours $\mathcal{T}_1$ and $\mathcal{T}_2$, and their cumulative cost can be computed in polynomial time. Hence, we route the vehicle according to the tours in $\mathcal{T}'$, where $\mathcal{T}' := \mathcal{T}_1$ if $Cu(\mathcal{T}_1) \leq Cu(\mathcal{T}_2)$ and $\mathcal{T}' := \mathcal{T}_2$ otherwise.

The details of $ALG.2(\lambda, \delta)$ is shown in Algorithm 3.

▶ **Theorem 13 (*).** *For Cu-VRPSD with any $\lambda \in (0, 1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$, conditioned on $\chi = d$, $ALG.2(\lambda, \delta)$ outputs a solution $\mathcal{T}$ with an expected cumulative cost of*

$$\frac{\gamma \cdot \left(\alpha \cdot \sigma + \int_0^\delta \frac{x}{\lambda - \delta} dF(x)\right) + \left(\frac{\lambda + \delta}{2} \cdot \alpha \cdot \sigma + \int_0^\delta \frac{(\lambda + \delta)x - x^2}{2(\lambda - \delta)} dF(x)\right)}{\gamma \cdot \max\{\sigma,\ 1\} + 0.5} \cdot LB + Cu(\mathcal{T}'),$$

where

$$Cu(\mathcal{T}') \leq \min\left\{ \frac{\int_\delta^1 \frac{2\gamma + x}{2} dF(x)}{\gamma \cdot \max\{\sigma,\ 1\} + 0.5} \cdot LB,\ \ \rho \cdot Cu(\mathcal{T}^*) \right\}.$$

3.2.1 The applications

Our goal is to obtain a 3.456-approximation algorithm for Cu-VRPSD with any $\gamma \in (1.444, \infty)$. As a byproduct, we will also get a 3.25-approximation algorithm for VRPSD.

Since in $APPROX.1(\lambda, \theta, p)$ $ALG.1(\lambda, \delta)$ sets $\lambda = 1$ for any $\gamma > 0.375$, we also set $\lambda = 1$ in $ALG.2(\lambda, \delta)$ for the sake of analysis. Moreover, since weighted 2-set cover [15] can be solved optimally in polynomial time, i.e., $\rho = 1$ when $\delta = 1/3$, we set $\delta = 1/3$ in $ALG.2(\lambda, \delta)$.

According to Theorems 9 and 13, we will show that $ALG.2(\lambda, \delta)$ can be used to make a trade-off with $ALG.1(\lambda, \delta)$. We use the approximation algorithm for Cu-VRPSD shown in Algorithm 4, denoted as $APPROX.2$.

▶ **Theorem 14 (*).** *For unsplittable Cu-VRPSD, $APPROX.2$ is a randomized 3.456-approximation algorithm for any $\gamma \in (1.444, \infty)$. Moreover, for unsplittable VRPSD, $APPROX.2$ is a randomized 3.25-approximation algorithm.*

Combining the results in Lemma 5, Theorems 11 and 14, we get the following result.

Algorithm 3 An algorithm for unsplittable Cu-VRPSD ($ALG.2(\lambda, \delta)$).

```
Input: An instance of unsplittable Cu-VRPSD, and two parameters $\lambda \in (0,1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$.
Output: A feasible solution $\mathcal{T}$ to unsplittable Cu-VRPSD.
Obtain an $\alpha$-approximate TSP tour $T^* = v_0v_1v_2 \ldots v_nv_0$, as Step 1 in $ALG.1(\lambda, \delta)$.
Load the vehicle with $S_0 := (L_0, \delta)$ units of goods, including $L_0$ demand of normal goods and $\delta$
  demand of backup goods, where $L_0 \sim U[0, \lambda - \delta)$.
Initialize $i := 1$ and $V^* := \emptyset$.
while $i \leq n$ do
    Go to customer $v_i$;
    if $\delta < d_i \leq 1$ then
        Record $v_i$'s demand, and let $V^* := V^* \cup \{v_i\}$ and $S_i := (L_i, \delta)$, where $L_i := L_{i-1}$;
    else if $d_i \leq L_{i-1}$ then
        Deliver $(d_i, 0)$ units of goods to $v_i$, and update $S_i := (L_i, \delta)$, where $L_i := L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot$
  $(\lambda - \delta) - d_i = L_{i-1} - d_i$;
    else                                        ▷ Since $d_i \leq \delta$, we must have $L_{i-1} < d_i \leq L_{i-1} + \delta$
        Deliver $(L_{i-1}, d_i - L_{i-1})$ units of goods to $v_i$, goes to the depot, load the vehicle with
  $(L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i, d_i - L_{i-1})$ units of goods, and update $S_i := (L_i, \delta)$, where
  $L_i := L_{i-1} + \lceil \frac{d_i - L_{i-1}}{\lambda - \delta} \rceil \cdot (\lambda - \delta) - d_i = L_{i-1} + (\lambda - \delta) - d_i$;
    end if
    $i := i + 1$.
end while
Go to the depot.
Consider a set of tours $\mathcal{T}_1$ by obtaining a single tour as in Step 22 for each $v_i \in V^*$.
Consider a set of tours $\mathcal{T}_2$ by calling a $\rho$-approximation algorithm for weighted $\frac{1-\delta}{\delta}$-set cover [15],
  where the instance is constructed as follows:
  1. Obtain all possible feasible sets $S$ of customers in $V^*$ such that the total demand of all
     customers in $S$ is at most 1;
  2. For each feasible set $S$, compute a tour with a minimum cumulative cost $Cu(S)$ for all
     customers in $S$;
  3. Get an instance of weighted $\frac{1-\delta}{\delta}$-set cover by taking each customer in $V^*$ as an element, and
     each feasible set $S$ as a weighted set with a weight of $Cu(S)$.
Let $\mathcal{T}' := \mathcal{T}_1$ if $Cu(\mathcal{T}_1) \leq Cu(\mathcal{T}_2)$ and $\mathcal{T}' := \mathcal{T}_2$ otherwise.
Route the vehicle according to the tours in $\mathcal{T}'$.
```

Algorithm 4 An approximation algorithm for unsplittable Cu-VRPSD ($APPROX.2$).

```
Input: An instance of unsplittable Cu-VRPSD.
Output: A feasible solution to Cu-VRPSD.
Call $ALG.1(1, 1/3)$ with a probability of 0.5 and call $ALG.2(1, 1/3)$ with a probability of 0.5.
```

▶ **Corollary 15.** *There is a randomized* 3.456*-approximation for unsplittable Cu-VRPSD.*

▶ Remark 16. We believe that our analysis is not tight. One one hand, it would be interesting to sharpen our analysis to get a better result; on the other hand, we may use $ALG.1$ and $ALG.2$ to design better approximation algorithms, e.g., with a probability of p_γ to run $ALG.1$, and of $(1 - p_\gamma)$ to run $ALG.2$, where p_γ is a function related to γ. Moreover, when running $ALG.1$ or $ALG.2$, the parameters λ and δ may follow a distribution related to γ.

4 Two Algorithms for Cu-VRP

In this section, we give a 3.194-approximation algorithm for Cu-VRP.

4.1 The first algorithm

Based on the well-known randomized rounding method for weighted k-set cover, we propose a 3.194-approximation algorithm, denoted as $ALG.3(\lambda, \delta)$, for Cu-VRP with any $\gamma > \gamma_0$, where $\gamma_0 > 0$ is any fixed constant.

Recall that $ALG.2(\lambda, \delta)$ first satisfies customers v_i with $d_i \leq \delta$ by traveling along the TSP tour and then customers v_i with $d_i > \delta$ by possibly solving weighted $\frac{1-\delta}{\delta}$-set cover. However, it may only be used for $\delta = 1/3$ since the best-known approximation ratio of weighted 3-set cover is about 1.79 [15], which is already too large.

In $ALG.3(\lambda, \delta)$, since the demands of customers are known in advance for Cu-VRP, we first try to satisfy customers in $V^* := \{v_i \in V' \mid d_i > \delta\}$ by solving weighted $\frac{1-\delta}{\delta}$-set cover using the randomized rounding method. Due to the randomness, some customers in V^* may still be unsatisfied. Then, we satisfy all remaining customers by calling $ALG.1(\lambda, \delta)$. This method was used to get an $(\alpha + 1 + \ln 2 + \varepsilon)$-approximation algorithm with any constant $\varepsilon > 0$ for unsplittable VRP [8]. The details are shown as follows.

To get an instance of weighted $\frac{1-\delta}{\delta}$-set cover, we use the method in Step 17 of $ALG.2(\lambda, \delta)$. Now, we have obtained a set of feasible sets $\mathcal{S}$, and each $S \in \mathcal{S}$ has a weight of $Cu(S)$. Then, we get the linear relaxation of weighted set cover as shown in (4), and it can be solved in $n^{O(1/\delta)}$ since $|\mathcal{S}| = n^{O(1/\delta)}$. In the randomized rounding method, we select each $S \in \mathcal{S}$ with a probability of $\min\{\ln 2 \cdot x_S, 1\}$. Denote the set of selected sets by $\mathcal{S}'$, which corresponds to a set of tours $\mathcal{T}'$ satisfying a subset of customers $V^{**} \subseteq V^*$. Note that $Cu(\mathcal{T}') \leq Cu(\mathcal{S}')$ since we may perform shortcutting to ensure that each customer appears in only one tour and it does not increase the cumulative cost by the triangle inequality. At last, we call $ALG.1(\lambda, \delta)$ to obtain a set of tours $\mathcal{T}''$ to satisfy the left customers in $V' \setminus V^{**}$. Due to the stochastic demands in Cu-VRPSD the load of the vehicle may be greater than the delivered units of goods in each tour of $\mathcal{T}''$. In Cu-VRP, we can optimize the tours in $\mathcal{T}''$ so that the load equals the delivered units of goods. Moreover, for each tour $T = v_0 v_{i_1} \dots v_{i_T} v_0 \in \mathcal{T}''$, we consider another tour with the opposite direction, i.e., $v_0 v_{i_T} \dots v_{i_1} v_0$, and choose the better one into our final solution.

$$\begin{aligned} \text{minimize} \quad & \sum_{S \in \mathcal{S}} Cu(S) \cdot x_S \\ \text{subject to} \quad & \sum_{S \in \mathcal{S} : v \in S} x_S \geq 1, \quad \forall\, v \in V^*, \\ & x_S \geq 0, \quad \forall\, S \in \mathcal{S}. \end{aligned} \tag{4}$$

The details of $ALG.3(\lambda, \delta)$ is shown in Algorithm 5.

▶ **Theorem 17 (*).** *For Cu-VRP with any $\lambda \in (0, 1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$, $ALG.3(\lambda, \delta)$ generates a solution $\mathcal{T}$ with an expected cumulative cost of*

$$\ln 2 \cdot Cu(\mathcal{T}^*) + \frac{\gamma \cdot \left(\alpha \cdot \sigma + \frac{1}{\lambda - \delta}\right) + \frac{\lambda}{2} \cdot \left(\alpha \cdot \sigma + \frac{1}{\lambda - \delta}\right)}{\gamma \cdot \max\{\sigma,\ 1\} + 0.5} \cdot LB.$$

▶ **Theorem 18 (*).** *For unsplittable Cu-VRP with any constants $\gamma_0 > 0$ and $\varepsilon > 0$, there is a randomized $(\alpha + 1 + \ln 2 + \varepsilon < 3.194)$-approximation algorithm for $\gamma > \gamma_0$.*

4.2 The second algorithm

In this section, we propose a 3.163-approximation algorithm for Cu-VRP with $\gamma \in (0, 0.428]$, denoted as $ALG.4(\lambda)$. Combing with Lemma 5 and Theorem 18, $ALG.4(\lambda)$ implies a 3.194-approximation algorithm for Cu-VRP.

Algorithm 5 An algorithm for unsplittable Cu-VRPSD ($ALG.3(\lambda, \delta)$).

```
Input: An instance of unsplittable Cu-VRPSD, and two parameters $\lambda \in (0, 1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$.
Output: A feasible solution $\mathcal{T}$ to unsplittable Cu-VRPSD.
1: Get an instance $(V^*, \mathcal{S})$ of weighted $\frac{1-\delta}{\delta}$-set cover using Step 17 in $ALG.2(\lambda, \delta)$.
2: Solve the linear program of weighted set cover in (4).
3: Select each $S \in \mathcal{S}$ with a probability of $\min\{\ln 2 \cdot x_S, 1\}$. Denote the set of selected sets by $\mathcal{S}'$, corresponding tours by $\mathcal{T}'$, and satisfied customers by $V^{**}$.
4: Call $ALG.1(\lambda, \delta)$ to obtain a set of tours $\mathcal{T}''$ to satisfy the customers in $V' \setminus V^{**}$.
5:   For each tour in $\mathcal{T}''$, ensure the load of the vehicle is the delivered units of goods, obtain another tour with the opposite direction, and choose the better one into $\mathcal{T}'''$.
6: Return $\mathcal{T}' \cup \mathcal{T}'''$.
```

Algorithm 6 An algorithm for unsplittable Cu-VRPSD ($ALG.4(\lambda)$).

```
Input: An instance of unsplittable Cu-VRPSD, and two parameters $\lambda \in (0, 1]$, $\delta \in (0, \lambda/2]$, and $1/\delta \in \mathbb{N}$.
Output: A feasible solution $\mathcal{T}$ to unsplittable Cu-VRPSD.
1: Call $ALG.1(\lambda, 0)$ to obtain a set of tours $\mathcal{T}'$ to satisfy all customers.
2:   For each tour in $\mathcal{T}'$, ensure the load of the vehicle is the delivered units of goods, obtain another tour with the opposite direction, and choose the better one into $\mathcal{T}$.
3: Return $\mathcal{T}$.
```

Algorithm 7 An approximation algorithm for unsplittable Cu-VRPSD ($APPROX.4(\lambda, \theta, p)$).

```
Input: An instance of unsplittable Cu-VRP, and three parameters $\lambda \in (0, 1]$, $\theta \in (0, 1)$ and $p \in (0, 1)$.
Output: A feasible solution to Cu-VRPSD.
1: Call $ALG.4(\lambda)$ with a probability of $p$ and call $ALG.4(\theta \cdot \lambda)$ with a probability of $1 - p$.
```

In $ALG.4(\lambda)$, we call $ALG.1(\lambda, 0)$ to obtain a set of tours $\mathcal{T}'$ to satisfy all customers, and then we optimize each tour in $\mathcal{T}'$ as Step 5 in $ALG.3$.

▶ **Theorem 19 (*).** *For Cu-VRP with any $\lambda \in (0, 1]$, $ALG.4(\lambda)$ generates a solution $\mathcal{T}$ with an expected cumulative cost of*

$$\frac{\gamma \cdot \left(\alpha \cdot \sigma + \int_0^\lambda \frac{2x}{\lambda} dF(x) + \int_\lambda^1 1 dF(x)\right) + \left(\frac{\lambda}{2} \cdot \alpha \cdot \sigma + \int_0^\lambda \frac{x^2/2 + \lambda \cdot x}{2\lambda} dF(x) + \int_\lambda^1 \frac{x}{2} dF(x)\right)}{\gamma \cdot \max\{\sigma,\ 1\} + 0.5} \cdot LB.$$

Similarly, we use $ALG.4(\lambda)$ to design an algorithm for Cu-VRP shown in Algorithm 7.

▶ **Theorem 20 (*).** *For unsplittable Cu-VRP, we can find (λ, θ, p) such that the approximation ratio of $APPROX.4(\lambda, \theta, p)$ is bounded by 3.163 for any $\gamma \in (0, 0.428]$.*

5 Conclusion

By using the idea of skipping customers with large demands during the TSP tour and satisfying them later, combined with careful analysis, we can improve the approximation ratio for Cu-VRPSD, VRPSD, and Cu-VRP. Whether this idea is also useful in designing practical algorithms for these problems is worthy of further study.

References

1 Kemal Altinkemer and Bezalel Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

2 Dimitris J. Bertsimas. A vehicle routing problem with stochastic demand. *Oper. Res.*, 40(3):574–585, 1992. doi:10.1287/opre.40.3.574.

3 Dimitris J. Bertsimas and David Simchi-Levi. A new generation of vehicle routing research: Robust algorithms, addressing uncertainty. *Oper. Res.*, 44(2):286–304, 1996. doi:10.1287/opre.44.2.286.

4 Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. *Math. Program.*, 197(2):451–497, 2023. doi:10.1007/s10107-022-01841-4.

5 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University, 1976.

6 Karina Corona-Gutiérrez, Samuel Nucamendi-Guillén, and Eduardo Lalla-Ruiz. Vehicle routing with cumulative objectives: A state of the art and analysis. *Comput. Ind. Eng.*, 169:108054, 2022. doi:10.1016/j.cie.2022.108054.

7 George B Dantzig and John H Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

8 Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Improved approximations for capacitated vehicle routing with unsplittable client demands. In Karen I. Aardal and Laura Sanità, editors, *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2022. doi:10.1007/978-3-031-06901-7_19.

9 Ricardo Fukasawa, Qie He, and Yongjia Song. A branch-cut-and-price algorithm for the energy minimization vehicle routing problem. *Transp. Sci.*, 50(1):23–34, 2016. doi:10.1287/trsc.2015.0593.

10 Daya Ram Gaur, Apurva Mudgal, and Rishi Ranjan Singh. Routing vehicles to minimize fuel consumption. *Oper. Res. Lett.*, 41(6):576–580, 2013. doi:10.1016/j.orl.2013.07.007.

11 Daya Ram Gaur, Apurva Mudgal, and Rishi Ranjan Singh. Approximation algorithms for cumulative VRP with stochastic demands. In Sathish Govindarajan and Anil Maheshwari, editors, *Algorithms and Discrete Applied Mathematics - Second International Conference, CALDAM 2016, Thiruvananthapuram, India, February 18-20, 2016, Proceedings*, volume 9602 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 2016. doi:10.1007/978-3-319-29221-2_15.

12 Daya Ram Gaur, Apurva Mudgal, and Rishi Ranjan Singh. Improved approximation algorithms for cumulative VRP with stochastic demands. *Discret. Appl. Math.*, 280:133–143, 2020. doi:10.1016/j.dam.2018.01.012.

13 Daya Ram Gaur and Rishi Ranjan Singh. A heuristic for cumulative vehicle routing using column generation. *Discret. Appl. Math.*, 228:140–157, 2017. doi:10.1016/j.dam.2016.05.030.

14 Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *Eur. J. Oper. Res.*, 88(1):3–12, 1996.

15 Anupam Gupta, Euiwoong Lee, and Jason Li. A local search-based approach for set covering. In Telikepalli Kavitha and Kurt Mehlhorn, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 1–11. SIAM, 2023. doi:10.1137/1.9781611977585.ch1.

16 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Technical note - approximation algorithms for VRP with stochastic demands. *Oper. Res.*, 60(1):123–127, 2012. doi:10.1287/opre.1110.0967.

17 Mordecai Haimovich and Alexander H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Math. Oper. Res.*, 10(4):527–542, 1985. doi:10.1287/moor.10.4.527.

18 İmdat Kara, Bahar Yetiş Kara, and M Kadri Yetis. Energy minimizing vehicle routing problem. In *Combinatorial Optimization and Applications: First International Conference, COCOA 2007*, pages 62–71. Springer, 2007.

19 İmdat Kara, Bahar Yetiş Kara, and M Kadri Yetis. *Cumulative vehicle routing problems*. IntechOpen Rijeka, HR, 2008.

20 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 32–45. ACM, 2021. doi:10.1145/3406325.3451009.

21 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A deterministic better-than-3/2 approximation algorithm for metric TSP. In Alberto Del Pia and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization - 24th International Conference, IPCO 2023, Madison, WI, USA, June 21-23, 2023, Proceedings*, volume 13904 of *Lecture Notes in Computer Science*, pages 261–274. Springer, 2023. doi:10.1007/978-3-031-32726-1_19.

22 Mauro Henrique Mulati, Ricardo Fukasawa, and Flávio Keidi Miyazawa. The arc-item-load and related formulations for the cumulative vehicle routing problem. *Discret. Optim.*, 45:100710, 2022. doi:10.1016/j.disopt.2022.100710.

23 Bahri Sahin, Huseyin Yilmaz, Yasin Ust, Ali Fuat Guneri, and Bahadir Gülsün. An approach for analysing transportation costs and a case study. *Eur. J. Oper. Res.*, 193(1):1–11, 2009. doi:10.1016/j.ejor.2007.10.030.

24 Anatolii Ivanovich Serdyukov. Some extremal bypasses in graphs. *Upravlyaemye Sistemy*, 17:76–79, 1978.

25 Xinyu Wang, Tsan-Ming Choi, Haikuo Liu, and Xiaohang Yue. A novel hybrid ant colony optimization algorithm for emergency transportation problems during post-disaster scenarios. *IEEE Trans. Syst. Man Cybern. Syst.*, 48(4):545–556, 2016.

26 Yuanxiao Wu and Xiwen Lu. Capacitated vehicle routing problem on line with unsplittable demands. *J. Comb. Optim.*, 44(3):1953–1963, 2022. doi:10.1007/s10878-020-00565-5.

27 Yiyong Xiao, Qiuhong Zhao, Ikou Kaku, and Yuchun Xu. Development of a fuel consumption optimization model for the capacitated vehicle routing problem. *Comput. Oper. Res.*, 39(7):1419–1431, 2012. doi:10.1016/j.cor.2011.08.013.